35th International Conference on Principles of Diagnosis and Resilient Systems

DX 2024, November 4–7, 2024, Vienna, Austria

Edited by

Ingo Pill
Avraham Natan
Franz Wotawa



Editors

Ingo Pill
Institute of Software Technology, Graz University of Technology, Austria
ingo.pill@gmail.com

Avraham Natan
Ben-Gurion University of the Negev, Israel
natanavr@post.bgu.ac.il

Franz Wotawa
Institute of Software Technology, Graz University of Technology, Austria
wotawa@ist.tugraz.at

ACM Classification 2012
Computing methodologies → Causal reasoning and diagnostics

ISBN 978-3-95977-356-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-356-0.

Publication date
November, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASIcs.DX.2024.0

ISBN 978-3-95977-356-0 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface
Ingo Pill, Avraham Natan, and Franz Wotawa 0:ix–0:x

Conference Organization
..... 0:xi

Authors
..... 0:xiii–0:xvi

Regular Papers

A Hierarchical Monitoring and Diagnosis System for Autonomous Robots
Gerald Steinbauer-Wagner, Leo Fürbaß, Marco De Bortoli, and Louise Travé-Massuyès 1:1–1:9

A Model-Based Approach for Monitoring and Diagnosing Digital Twin Discrepancies
Elaheh Hosseinkhani, Martin Leucker, Martin Sachenbacher, Hendrik Streichhahn, and Lars B. Vosteen 2:1–2:15

A Review of Fault Diagnosis Techniques Applied to Aircraft Air Data Sensors
Lucas Lima Lopes, Louise Travé-Massuyès, Carine Jauberthie, and Guillaume Alcalay 3:1–3:20

A Study on Redundancy and Intrinsic Dimension for Data-Driven Fault Diagnosis
Daniel Jung and David Axelsson 4:1–4:17

Bridging Hardware and Software Diagnosis: Leveraging Fault Signature Matrix and Spectrum-Based Fault Localization Similarities
Louise Travé-Massuyès and Franz Wotawa 5:1–5:15

Challenges for Model-Based Diagnosis
Ingo Pill and Johan de Kleer 6:1–6:20

Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research
Daniel Vranješ, Jonas Ehrhardt, René Heesch, Lukas Moddemann, Henrik Sebastian Steude, and Oliver Niggemann 7:1–7:13

Diagnosing Multi-Agent STRIPS Plans
Avraham Natan, Roni Stern, Meir Kalech, William Yeoh, and Tran Cao Son 8:1–8:20

Inferring Sensor Placement Using Critical Pairs and Satisfiability Modulo Theory
Alexander Diedrich, René Heesch, Marco Bozzano, Björn Ludwig, Alessandro Cimatti, and Oliver Niggemann 9:1–9:19

Leveraging Answer Set Programming for Continuous Monitoring, Fault Detection, and Explanation of Automated and Autonomous Driving Systems
Lorenz Klampfl and Franz Wotawa 10:1–10:20

35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Leveraging Causal Information for Multivariate Timeseries Anomaly Detection
Lukas Heppel, Andreas Gerhardus, Ferdinand Rewicki, Jan Deeken, and Günther Waxenegger-Wilfing 11:1–11:18

Minimalist Diagnosis of Discrete-Event Systems
Gianfranco Lamperti and Marina Zanella 12:1–12:20

MSO Sets and MTES for Dummies
Maxence Glotin, Louise Travé-Massuyès, and Elodie Chanthery 13:1–13:15

One-Class Classification and Cluster Ensembles for Anomaly Detection and Diagnosis in Multivariate Time Series Data
Adil Mukhtar, Thomas Hirsch, and Gerald Schweiger 14:1–14:19

Property Learning-Based Fault Detection for Liquid Propellant Rocket Engine Control Systems
Andrea Urgolo, Ingo Pill, Günther Waxenegger-Wilfing, and Manuel Freiberger ... 15:1–15:20

Quantifying the Sim-To-Real Gap in UAV Disturbance Rejection
Austin Coursey, Marcos Quinones-Grueiro, and Gautam Biswas 16:1–16:18

Real-Time Sensor Fault Detection in Drones: A Correlation-Based Algorithmic Approach
Inbal Roshanski, Magenya Roshanski, and Meir Kalech 17:1–17:20

Simulation-Based Diagnosis for Cyber-Physical Systems – A General Approach and Case Study on a Dual Three-Phase E-Machine
David Kaufmann, Matus Kozovsky, and Franz Wotawa 18:1–18:21

Short Papers

Achieving Complete Structural Test Coverage in Embedded Systems Using Trace-Based Monitoring
Alexander Weiss, Albert Schulz, Martin Heininger, Martin Sachenbacher, and Martin Leucker 19:1–19:12

Data-Driven Diagnosis of Electrified Vehicles: Results from a Structured Literature Review
Stan Muñoz Gutiérrez, Adil Mukhtar, and Franz Wotawa 20:1–20:14

Data-Driven RUL Prediction Using Performance Metrics
Abel Diaz-Gonzalez, Austin Coursey, Marcos Quinones-Grueiro, Chetan S. Kulkarni, and Gautam Biswas 21:1–21:15

Detecting Soft Faults in Heat Pumps
Birgit Hofer and Franz Wotawa 22:1–22:10

Diagnosing Non-Intermittent Anomalies in Reinforcement Learning Policy Executions
Avraham Natan, Roni Stern, and Meir Kalech 23:1–23:13

Faster Diagnosis with Answer Set Programming
Liliana Marie Prikler and Franz Wotawa 24:1–24:13

FLEX: Fault Localization and Explanation Using Open-Source Large Language Models in Powertrain Systems
Herbert Muehlburger and Franz Wotawa 25:1–25:14

Hyperplanes Based Zonotopic Contractor
Rahma Bengamra, Soheib Fergani, and Carine Jauberthie 26:1–26:13

On a Method to Measure Supervised Multiclass Model's Interpretability: Application to Degradation Diagnosis
Charles-Maxime Gauriat, Yannick Pencolé, Pauline Ribot, and Gregory Brouillet . 27:1–27:14

Test Selection for Diagnosing Multimode Systems
Mattias Krysander and Fatemeh Hashemniya 28:1–28:14

Transformer-Based Signal Inference for Electrified Vehicle Powertrains
Stan Muñoz Gutiérrez, Adil Mukhtar, and Franz Wotawa 29:1–29:14

Usability of Symbolic Regression for Hybrid System Identification – System Classes and Parameters
Swantje Plambeck, Maximilian Schmidt, Audine Subias, Louise Travé-Massuyès, and Goerschwin Fey 30:1–30:14

Using Multi-Modal LLMs to Create Models for Fault Diagnosis
Silke Merkelbach, Alexander Diedrich, Anna Sztyber-Betley, Louise Travé-Massuyès, Elodie Chanthery, Oliver Niggemann, and Roman Dumitrescu 31:1–31:15

Extended Abstracts

Summary of "A Lazy Approach to Neural Numerical Planning with Control Parameters"
René Heesch, Alessandro Cimatti, Jonas Ehrhardt, Alexander Diedrich, and Oliver Niggemann 32:1–32:3

Summary of "Randomized Problem-Relaxation Solving for Over-Constrained Schedules"
Patrick Rodler, Erich Teppan, and Dietmar Jannach 33:1–33:4

Summary of "Sequence-Oriented Diagnosis of Discrete-Event Systems"
Gianfranco Lamperti, Stefano Trerotola, Marina Zanella, and Xiangfu Zhao 34:1–34:2

Preface

In 2024, the 35th International Conference on Principles of Diagnosis and Resilient Systems (DX) took place from November 4th to 7th in Vienna, Austria. For this DX edition, we saw two significant changes from previous ones. That is, for 35 years, DX has been known as the International Workshop on Principles of Diagnosis, but starting with the 2024 edition, DX is a full conference with formal open-access proceedings. Furthermore, we widened our primary focus to include also resilient systems – a research area that is tightly interconnected with research on diagnosis.

The DX conference is a premier forum to present and discuss the latest research, experience reports, and emerging ideas in the context of diagnosis and resilient systems. DX is application-agnostic, so that the presented research is related to a large variety of systems, ranging from physical to computational ones, captured in abstract or detailed representations. In previous DX iterations, the main focus was on diagnosis, i.e., the identification of root causes for encountered issues and unexpected scenarios, and on related techniques such as prognostics, planning, and control. Moving forward, we expanded our focus on diagnosis to include the topic of resilience, which is the intrinsic ability of a system to sustain its required operations when impacted by expected and unexpected contingencies. DX welcomes papers that cover resilient design as well as approaches for operational resilience. Topics covered by the DX conference include but are not limited to:

- Monitoring, detection, diagnosis, and mitigation of faults, unexpected issues, and change
- Formal theories and symbolic, sub-symbolic, as well as hybrid approaches for diagnosis
- Connections between diagnosis and other techniques like decision making, (re-)planning & (re-)configuration, control theory, formal verification, and testing
- Concept papers on the theory of design and operational resilience
- Designing, developing, and operationalizing resilient systems
- Hard- and software instrumentation, as well as dependable data acquisition and probing
- Development, learning, abstraction, transformation, analysis, optimization, and transfer of diagnosis models
- Diagnosis in resilient, intelligent, and autonomous systems
- Diagnosis in a distributed, hierarchical, system-of-systems, or multi-agent context

For DX'24, we received 45 submissions in total, despite the organizational delays due to the relocation from Israel. Out of these 45, 42 papers underwent the reviewing process. 18 (40 %) were finally accepted as full papers, 13 (28.9 %) as short papers, and three submissions for the journal-first/major-conference-first track were accepted as extended abstracts.

The papers in these proceedings perfectly illustrate the diversity of research related to diagnosis and resilient systems. Their foci range from state estimation, via applications like robotics, electric motors, and heat pumps, to addressing fundamental questions. Examples for the latter are related to the integration of approaches from different fields, the application of the newest findings in other areas like LLMs for the benefit of diagnosis and resilient systems, or discussing upcoming challenges for the DX research of the future.

Organizing a conference is always a group effort, so that we would like to thank a few people, starting with Elisabeth Orthofer and Alexander Perko from our local support team from the Institute of Software Technology of the Graz University of Technology.

35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We would also like to thank the members of the DX steering committee that has been directing the transition of DX becoming a conference since it was founded in 2023:

- Ingo Pill (chair), Graz University of Technology, Austria
- Gautam Biswas, Vanderbilt university, USA
- Johan de Kleer, c-infinity, USA
- Meir Kalech, Ben Gurion University of the Negev, Israel
- Oliver Niggemann, Helmut-Schmidt-Universität Hamburg, Germany
- Louise Travé-Massuyès, LAAS-CNRS, France
- Franz Wotawa, Graz University of Technology, Austria
- Marina Zanella, Università di Brescia, Italy

Our thanks go also to the Graz University of Technology and the Ben Gurion University of the Negev for supporting the organizers in their efforts to make the conference happen. Special thanks go to the AI Journal for sponsoring DX'24 and to the members of the program committee for their reviews that allowed us to single out the right papers to constitute a high-quality technical program for the 35th International Conference on Principles of Diagnosis and Resilient Systems. We would furthermore like to thank Dagstuhl, our publisher, for working with us to make the technical program available via open-access proceedings.

Finally, we would like to thank all the paper authors for contributing to the success of this conference. Without their work and their choice to present it at DX, this event would not have been possible.

The DX'24 chairs: Ingo Pill, Avraham Natan, and Franz Wotawa.

Conference Organization

The DX'24 program committee (in a.o.):

- Rui Abreu, INESC-ID & Universidade do Porto, Portugal
- Carlos Alonso-Gonzalez, Universidad de Valladolid, Spain
- Gautam Biswas, Vanderbilt University, USA
- Elodie Chanthery, LAAS-CNRS, France
- Alessandro Cimatti, FBK, Italy
- Johan DeKleer, SRI International, USA
- Alexander Diedrich, Helmut-Schmidt-University, Germany
- Gerhard Friedrich, AAU Klagenfurt, Austria
- Daniel Jung, Linköping University, Sweden
- Meir Kalech, Ben Gurion Univ. of the Negev, Israel
- Bruno Lima, University of Porto, Portugal
- Angelo Montanari, University of Udine, Italy
- Avraham Natan, Ben-Gurion University, Israel (Chair)
- Oliver Niggemann, HSU-HH, Germany
- Yannick Pencolé, LAAS-CNRS, France
- Ingo Pill, TU Graz, Austria (Chair)
- Gregory Provan, University College Cork, Ireland
- Belarmino Pulido, Universidad de Valladolid, Spain
- Pauline Ribot, LAAS-CNRS, France
- Patrick Rodler, AAU Klagenfurt, Austria
- Indranil Roychoudhury, Schlumberger, USA
- Martin Sachenbacher, Univ. Lübeck, Germany
- Anika Schumann, IBM Research Zurich, Switzerland
- Neal Snooke, Aberystwyth University, UK
- Gerald Steinbauer-Wagner, TU Graz, Austria
- Peter Struss, TU München, Germany
- Markus Stumptner, Univ. of South Australia, Australia
- Anna Sztyber, Warzaw Univ. o. Technology, Poland
- Stefano Tonetta, FBK, Italy
- Gianluca Torta, Università di Torino, Italy
- Louise Travé-Massuyès, LAAS-CNRS, France
- Franz Wotawa, TU Graz, Austria (Chair)
- Marina Zanella, Università degli Studi di Brescia, Italy

35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Guillaume Alcalay
Airbus Operations S.A.S
Toulouse, France
guillaume.alcalay@airbus.com

Luis Alvarez
MIT Lincoln Laboratory
Lexington, MA, United States
luis.alvarez@ll.mit.edu

David Axelsson
Linköping University
Linköping, Sweden
davidaxelsson5@gmail.com

Rahma Bengamra
LAAS
Toulouse, France
rahma.bengamra@laas.fr

Gautam Biswas
Vanderbilt University
Nashville, TN, United States
gautam.biswas@vanderbilt.edu

Marco Bozzano
Fondazione Bruno Kessler
Povo, Italy
bozzano@fbk.eu

Gregory Brouillet
BOSCH France
France
gregory.brouillet@fr.bosch.com

Elodie Chanthery
LAAS-CNRS, University of Toulouse
Toulouse, France
elodie.chanthery@laas.fr

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, Italy
cimatti@fbk.eu

Austin Coursey
Vanderbilt University
Nashville, TN, United States
austin.c.coursey@vanderbilt.edu

Marco De Bortoli
Graz University of Technology
Graz, Austria
mbortoli@ist.tugraz.at

Johan de Kleer
c-infinity
United States
johan@dekleer.org

Jan Deeken
German Aerospace Center (DLR)
Hardthausen am Kocher, Germany
jan.deeken@dlr.de

Abel Diaz-Gonzalez
Vanderbilt University
Nashville, TN, United States
abel.diaz.gonzalez@vanderbilt.edu

Alexander Diedrich
Helmut Schmidt University
Hamburg, Germany
alexander.diedrich@hsu-hh.de

Roman Dumitrescu
Paderborn University
Paderborn, Germany
roman.dumitrescu@iem.fraunhofer.de

Jonas Ehrhardt
Helmut-Schmidt University
Hamburg, Germany
jonas.ehrhardt@hsu-hh.de

Soheib Fergani
LAAS
Toulouse, France
soheib.fergani@laas.fr

Goerschwin Fey
Hamburg University of Technology
Hamburg, Germany
goerschwin.fey@tuhh.de

Manuel Freiberger
Silicon Austria Labs (SAL)
Graz, Austria
manuel.freiberger@silicon-austria.com

35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Leo Fürbaß
Graz University of Technology
Graz, Austria
leo.fuerbass@student.tugraz.at

Charles-Maxime Gauriat
LAAS-CNRS, BOSCH France
Toulouse, France
charlesmaxime.gaurait@fr.bosch.com

Andreas Gerhardus
German Aerospace Center (DLR)
Hardthausen am Kocher, Germany
andreas.gerhardus@dlr.de

Maxence Glotin
LAAS-CNRS, Université de Toulouse
Toulouse, France
maxence.glotin@laas.fr

Fatemeh Hashemniya
Linköping University
Linköping, Sweden
Fatemeh.Hashemniya@liu.se

René Heesch
Helmut-Schmidt University
Hamburg, Germany
rene.heesch@hsu-hh.de

Lukas Heppel
University of Würzburg, German Aerospace Center (DLR)
Würzburg, Germany
lukasheppel00@gmail.com

Thomas Hirsch
Graz University of Technology
Graz, Austria
thirsch@ist.tugraz.at

Birgit Hofer
Graz University of Technology
Graz, Austria
bhofer@ist.tugraz.at

Elaheh Hosseinkhani
University of Lübeck
Lübeck, Germany
hosseinkhani@isp.uni-luebeck.de

Dietmar Jannach
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
dietmar.jannach@aau.at

Carine Jauberthie
LAAS-CNRS, Université de Toulouse, UPS
Toulouse, France
carine.jauberthie@laas.fr

Daniel Jung
Linköping University
Linköping, Sweden
daniel.jung@liu.se

Meir Kalech
Ben-Gurion University of the Negev
Beer-Sheva, Israel
kalech@bgu.ac.il

David Kaufmann
Graz University of Technology
Graz, Austria
david.kaufmann@ist.tugraz.at

Lorenz Klampfl
Graz University of Technology
Graz, Austria
lklampfl@ist.tugraz.at

Matus Kozovsky
Brno University of Technology
Brno, Czechia
matus.kozovsky@ceitec.vutbr.cz

Mattias Krysander
Linköping University
Linköping, Sweden
mattias.krysander@liu.se

Chetan S. Kulkarni
NASA Ames Research Center
Mountain View, CA, United States
chetan.s.kulkarni@nasa.gov

Gianfranco Lamperti
University of Brescia
Brescia, Italy
gianfranco.lamperti@unibs.it

Martin Leucker
University of Lübeck
Lübeck, Germany
leucker@isp.uni-luebeck.de

Lucas Gabriel Lima Lopes
Airbus Operations S.A.S, LAAS-CNRS
Toulouse, France
lucas-gabriel.lima-lopes@airbus.com

Björn Ludwig
Helmut Schmidt University
Hamburg, Germany
ludwigb@hsu-hh.de

Silke Merkelbach
Fraunhofer IEM
Paderborn, Germany
silke.merkelbach@iem.fraunhofer.de

Lukas Moddemann
Helmut-Schmidt University
Hamburg, Germany
lukas.moddemann@hsu-hh.de

Herbert Mühlburger
Graz University of Technology
Graz, Austria
muehlburger@ist.tugraz.at

Adil Mukhtar
Graz University of Technology
Graz, Austria
amukhtar@ist.tugraz.at

Stalin Muñoz Gutiérrez
Graz University of Technology
Graz, Austria
sgutierr@ist.tugraz.at

Avraham Natan
Ben-Gurion University of the Negev
Beer-Sheva, Israel
natanavr@post.bgu.ac.il

Oliver Niggemann
Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg
Hamburg, Germany
info@oliver-niggemann.de

Yannick Pencolé
LAAS-CNRS
Toulouse, France
yannick.pencole@laas.fr

Ingo Pill
Graz University of Technology
Graz, Austria
ingo.pill@gmail.com

Swantje Plambeck
Hamburg University of Technology
Hamburg, Germany
swantje.plambeck@tuhh.de

Liliana Marie Prikler
Graz University of Technology
Graz, Austria
liliana.prikler@ist.tugraz.at

Marcos Quinones-Grueiro
Vanderbilt University
Nashville, TN, United States
marcos.quinones.grueiro@vanderbilt.edu

Ferdinand Rewicki
German Aerospace Center (DLR)
Hardthausen am Kocher, Germany
ferdinand.rewicki@dlr.de

Pauline Ribot
LAAS-CNRS
Toulouse, France
pribot@laas.fr

Patrick Rodler
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
patrick.rodler@aau.at

Inbal Roshanski
Ben-Gurion University of the Negev
Beer-Sheva, Israel
inbalros@post.bgu.ac.il

Magenya Roshanski
Maris-Tech Ltd
Rehovot, Israel
mgr@maris-tech.com

Martin Sachenbacher
University of Lübeck
Lübeck, Germany
sachenba@googlemail.com

Maximilian Schmidt
Hamburg University of Technology
Hamburg, Germany
maximilian.schmidt@tuhh.de

Albert Schulz
Accemic Technologies GmbH
Kiefersfelden, Germany
aschulz@accemic.com

Gerald Schweiger
Vienna University of Technology
Vienna, Austria
gerald.schweiger@tuwien.ac.at

Tran Cao Son
New Mexico State University
Las Cruces, NM, United States
tson@cs.nmsu.edu

Gerald Steinbauer-Wagner
Graz University of Technology
Graz, Austria
steinbauer@ist.tugraz.at

Roni Stern
Ben-Gurion University of the Negev
Beer-Sheva, Israel
roni.stern@gmail.com

Henrik Sebastian Steude
Helmut-Schmidt-University
Hamburg, Germany
henrik.steude@hsu-hh.de

Hendrik Streichhahn
Dräger Safety AG & Co. KGaA.
Germany
hendrik.streichhahn@draeger.com

Audine Subias
LAAS-CNRS
Toulouse, France
subias@laas.fr

Anna Sztyber-Betley
Warsaw University of Technology
Warsaw, Poland
anna.sztyber@pw.edu.pl

Erich Teppan
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
erich@ifit.uni-klu.ac.at

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse
Toulouse, France
louise@laas.fr

Stefano Trerotola
University of Brescia
Brescia, Italy
stefano.trerotola@outlook.it

Andrea Urgolo
Silicon Austria Labs (SAL)
Graz, Austria
andrea.urgolo@silicon-austria.com

Lars Bernd Vosteen
University of Lübeck
Lübeck, Germany
lars.vosteen@isp.uni-luebeck.de

Daniel Vranješ
Helmut-Schmidt-University
Hamburg, Germany
daniel.vranjes@hsu-hh.de

Günther Waxenegger-Wilfing
German Aerospace Center (DLR)
Hardthausen am Kocher, Germany
guenther.waxenegger@dlr.de

Alexander Weiss
Accemic Technologies GmbH
Kiefersfelden, Germany
aweiss@accemic.com

Franz Wotawa
Graz University of Technology
Graz, Austria
wotawa@ist.tugraz.at

William Yeoh
Washington University in St. Louis
St. Louis, MO, United States
wyeoh@wustl.edu

Marina Zanella
University of Brescia
Brescia, Italy
marina.zanella@unibs.it

Xiangfu Zhao
Yantai University
Yantai, Shandong, China
xiangfuzhao@gmail.com

A Hierarchical Monitoring and Diagnosis System for Autonomous Robots

Gerald Steinbauer-Wagner
Institute for Software Technology, Graz University of Technology, Austria

Leo Fürbaß
Institute for Software, Technology Graz University of Technology, Austria

Marco De Bortoli
Institute for Software Technology, Graz University of Technology, Austria

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Abstract

This paper addresses the capability of autonomous robots to achieve flexible goals in dynamic environments. In such a setting numerous challenges jeopardize the robustness of such systems. Thus, we propose a hierarchical diagnosis concept for layered control architectures, that can detect and deal with such challenges to maintain a consistent knowledge about the world and to allow reliable decision-making. Layered control systems use various knowledge representations and decision-making mechanisms teamed with specialized isolated fault-handling approaches. However, some issues can only be identified if the information from different layers is combined. Our approach addresses challenges like failing actions, uncertain observations, and unmodeled events by propagating observations and diagnoses results throughout the hierarchy. This enhances adaptability and dependability in various domains. In this paper, we present a prototype architecture following this approach.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Cognitive Architecture, Autonomous Agents, Dependability, Hierarchical Monitoring and Diagnosis

Digital Object Identifier 10.4230/OASIcs.DX.2024.1

1 Introduction

There is an increasing demand for autonomous robots that can pursue flexible goals in an open and dynamic context. This asks for true autonomy, requiring minimal or no human intervention. In contrast, even commercial robot systems often fail to pursue their goals and interact with an unpredictable dynamic environment. There are various reasons for this shortcoming such as failing actions, not modeled or observed changes in the environment, or issues in the perception. In this paper, we propose a hierarchical diagnosis concept for a common control architecture for autonomous robots. Such architectures are usually organized in layers. When progressing from one layer up to the next usually information is abstracted and the scope of the decision-making is narrowed conceptually (e.g. aggregated predicates) and widened temporally (e.g. sequence of intermediate actions). Moreover, the higher layer uses the lower one to refine and execute its decisions. In this interplay information, commands, and execution results need to be exchanged while performing abstraction and concretization respectively (i.e. instantiation of parameter or aggregation of results). In order to allow dependable decision-making and execution the knowledge and information on the various levels needs to be kept consistent with the evolution of the environment [15]. A standard architecture is a three-tier architecture that combines a deliberative layer with


© Gerald Steinbauer-Wagner, Leo Fürbaß, Marco De Bortoli, and Louise Travé-Massuyès;
licensed under Creative Commons License CC-BY 4.0



planning capabilities, an executive layer able to execute intermediate tasks by refining them into skills (atomic actions), and a skill-layer able to handle the execution of such skills [12]. Monitoring and diagnosis need to be performed on the various levels in the corresponding layer context. The main challenges for an intelligent agent and its control architecture performing non-trivial tasks in dynamic environments are: (1) skills (actions) that are not able to establish all its intended effects, (2) observations about the environment that are uncertain or wrong, (3) exogenous events changing the environment in a way the agent has not modeled or is not aware of, and (4) failed or interrupted skills (actions) where the resulting situation is not entirely clear. All these challenges may render the local context of the various levels inconsistent jeopardizing dependable decision-making and execution. In order to address these challenges we propose a holistic hierarchical diagnosis concept that propagates local observations, execution results, and diagnoses up and down the control hierarchy to allow reasoning with a broadening context if issues cannot be detected or handled within the corresponding layer. The diagnosis concept presented in this paper augments our planning and execution framework, which was designed in a general way and can be easily adapted to different domains.

2 The Control Architecture

In this section, we briefly describe the general architecture of the control framework. It resembles the general structure used for multi-robot systems, and the main parts of the software are divided into a three-level hierarchy, as shown in Figure 1. The most abstract layer, namely the High-Level, involves task generation, task assignment, and coordination between robots. It is responsible for the long-term strategy and monitoring of its execution. The Mid-Level is concerned with task refinement and execution by a specific agent through Behavior Trees [7]. Given a High-Level action, this layer transforms it into the corresponding Behavior Tree. The Behavior Tree decomposes the actions into a set of subtasks, represented as an extended state machine. To give an insight, consider a high-level action of passing through a closed door. Such action will be transformed into a set of different subtasks by the Mid-Level. Passing to the door might be represented by a sequence of checking the door state, opening the door if needed, passing it, and finally closing it. This level of detail is neither useful nor helpful for the High-Level. The Low-Level deals with the basic behaviors of the robot platform, i.e., sensing, low-level monitoring and control of actuators as well as basic skills such as localization and navigation. Here the subtasks generated at the Mid-Level are further refined.

2.1 The High-Level Planning and Dispatching System

The Planning and Dispatching Framework implements a control strategy for multi-agent systems in dynamic domains. It is based on three main components: (1) a Goal Reasoner, (2) a Planner, and (3) a Dispatching and Monitoring system. In the upper part of Figure 1, the interaction between the components is shown. It follows the idea of Goal-Driven Autonomy. The Dispatching and Monitoring component plays the role of the main controller that invokes the Goal Reasoner [1], and consequently the planner, and executes the obtained plan, by sending the tasks to the Mid-Level which builds the corresponding Behavior Trees. The plan execution is constantly monitored for issues that may require regeneration of the goals of the plan, e.g., failed actions, deadline violations, or external events. The planner module is based on Temporal PDDL [11]. Given a fixed domain, namely the description

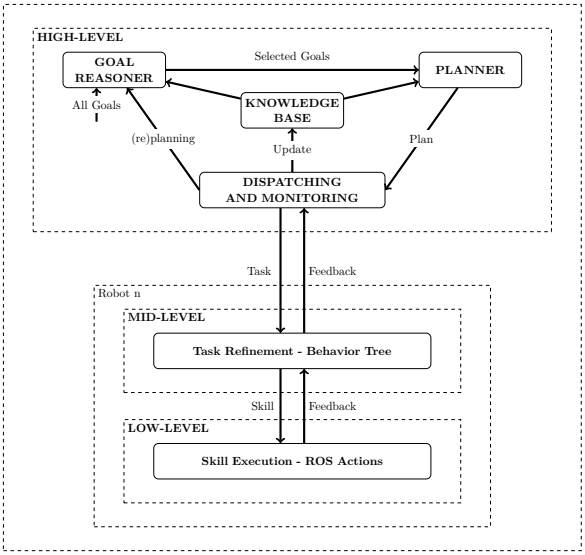


Figure 1 Basic hierarchical control architecture for multi-agent systems.

of possible actions, and the problem instance, generated from the Knowledge Base and the selected goals, the result of the planning process is a temporal plan, which is represented by the schedule σ, formed by a set of triples $\langle a, t_a, d_a \rangle$, where a is an action, t_a is the time when the action a needs to be started, and d_a is the duration of the action. The Knowledge Base, namely the set of beliefs about the status of the world, is a crucial component. For easier integration with PDDL planning, the Knowledge Base reflects PDDL syntax. It consists of a set of atoms. An atom is a grounded predicate. For instance, the atom $(at\ r\ l)$ specifies that the robot r is at location l. Each Temporal PDDL action of the plan features a set of preconditions and effects. The preconditions of an action a are formed by three sets of atoms: the start condition $cond_{\vdash}(a)$, the invariant condition $cond_{\leftrightarrow}(a)$, and the end condition $cond_{\dashv}(a)$. The effects are instead divided into the start positive (add) effects $eff^{+}_{\vdash}(a)$, the start negative (remove) effects $eff^{-}_{\vdash}(a)$, the end positive (add) effects $eff^{+}_{\dashv}(a)$ and the end negative (remove) effects $eff^{-}_{\dashv}(a)$. Every time an action is dispatched, the Knowledge Base KB is checked for consistency by making sure that $cond_{\vdash}(a) \in KB$ and $cond_{\leftrightarrow}(a) \in KB$. If it holds, we then apply the starting effect of the action, with $KB \leftarrow KB \cup eff^{+}_{\vdash}(a)$ and $KB \leftarrow KB \setminus eff^{-}_{\vdash}(a)$. The same procedure is applied for *end* actions, considering end conditions and effects instead. ROSPlan follows a similar approach [5]. As a consequence, in case of failures or mismatches between the Knowledge Base and the real world, it is crucial to diagnose the problem accurately and identify the actual effects on the belief. In fact, a Knowledge Base inconsistent with the state of the environment may compromise the entire decision-making. By using monitoring and diagnosis, it may be possible to track the specific problem in the interaction of the agent with its environment and fix the KB accordingly, allowing it to pursue its goals. Therefore, the diagnosis system presented in the next chapter has been developed.

2.2 Control and Recovery Strategy in the Mid-Level

The use of Behaviour Trees (BT) in Mid-Level not only allows realizing action refinement easily but also the integration with supervison and recovery strategies. Conditions, fallbacks, and other predefined types of nodes allow for an easy implementation of both supervision

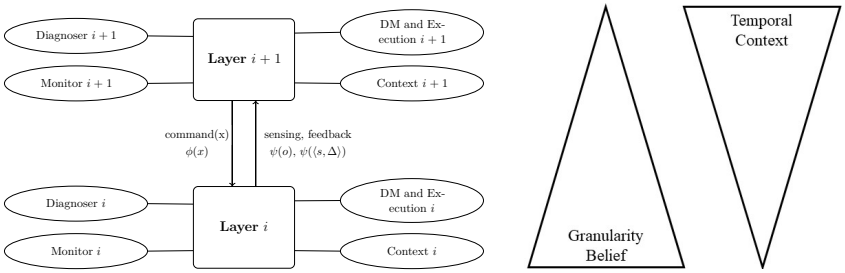


Figure 2 Hierarchical Organization of Layer Interaction, Monitoring, and Diagnosis.

and recovery. For this, the original BT that refines an action is transferred into a dependable failsafe version, where patterns for fault monitoring and mitigation are encoded using domain/expert knowledge. An example is the use of a ReactiveSequence node, to continuously check guarding conditions. Moreover, this pattern is a subtree of a BT having a RetryUntilSuccessful node as the root. This node retries to execute the whole subtree in case of failure, within a predefined maximum number of attempts. This allows the implementation of a simple local recovery strategy. If recovery fails on this level, the problem has to be handled either through local diagnosis or by the deliberative layer.

3 Hierarchical Monitoring and Diagnosis for Decision Making and Execution

The use of an architecture with various levels of abstraction, necessary to tackle complex dynamic domains requiring planning, synchronization, and skill execution, poses numerous challenges regarding the implementation of a holistic monitoring and diagnosis system. In order to correctly detect and identify problems in task planning and execution, the system must be aware of the different layers and heterogeneous knowledge representations and decision-making mechanisms used at the different layers. Moreover, it needs to understand how the different layers interact and whether there is already a local mechanism for fault detection and handling on some layers sufficient to deal with a problem.

Hierarchical diagnosis is a well-known concept, to speed up diagnosis in complex systems using functional abstraction and clustering [6]. Functional abstraction establishes a framework that maps structural and behavioral knowledge, typically found at the low-level, and teleological knowledge, which pertains to the system's goals at the high-level. Functional knowledge, situated at the mid-level, bridges the gap between these two types of knowledge. The core concept of a functional description is to link behavioral and teleological knowledge by identifying how the structural components contribute to fulfilling the system's overall goal, showcasing the functional roles these components play in achieving the desired goals. In our case, the challenge is to combine information and diagnosis results from different levels of the architecture into a globally consistent diagnosis. Functional abstraction provides the means to interpret the results of one level into the other levels. An interesting aspect regarding diagnosis is that the temporal context and the conceptual context show an inverse proportional nature along the levels. While the level of granularity of facts shrinks when moving up the hierarchy (abstraction of actions and beliefs), the temporal context becomes wider (a sequence of actions versus only one behavior).

In order to address these challenges, we propose a hierarchical monitoring and diagnosis system where the interaction between a layer $i+1$ and i follows the pattern shown in Figure 2. We assume that each layer i possesses its individual set of the following components: (1)

context, (2) decision-making and execution (DME), (3) monitoring, and (4) diagnosis. The implementation of the individual components depends on the layer. In the subsequent sections, we will provide details for the layers used in our architecture. The context holds all information relevant for decision-making and execution. We assume the context is represented using some language $\mathcal{L}_i$. DME can come up with a decision on what activities need to be performed to achieve a given goal or command and can execute that decision. The monitor supervises the correct execution of the decision while the diagnoser provides explanations of why an execution failed. DME might be able to use those diagnoses to reconfigure the decision and execution or propagate them to the higher layer if it is not able to handle it sufficiently.

In order to represent the interaction between layer $i+1$ and i we assume that layer $i+1$ issues commands represented in $\mathcal{L}_{i+1}$. For layer i we assume a function $\phi : \mathcal{L}_{i+1} \rightarrow \mathcal{L}_i$ which performs the concretization of that command. Layer i will work on the achievement of that command. During the handling of the command intermediate observations o represented in $\mathcal{L}_i$ will be propagated to layer $i+1$ that can use these observations for monitoring. In order to allow an integration of o in the context $i+1$ we assume an abstraction function $\psi : \mathcal{L}_i \rightarrow \mathcal{L}_{i+1}$. If the processing of the command concludes (nominal or abnormal) the feedback tuple $\langle s, \Delta \rangle$ where $s \in \{success, fail\}$ represents the execution result and Δ (represented in $\mathcal{L}_i$) potential explanations for a failed execution.

A reported failure occurs if a monitor at layer i detects a problem or if a lower layer's issue is propagated and unresolved. Even reported success may not mean correct completion, as issues might only be detectable in higher layers utilizing the broader context. Thus, higher-layer monitors must verify lower-layer execution results. For instance, in a production setting a skill to insert a piece into a machine may drop it without noticing it due to limited sensing but at the deliberative layer the issue can be revealed because the following processing action fails due to the missing implicit precondition of the inserted piece.

3.1 Skill Layer (Low-Level)

The lowest layer is responsible for the execution of atomic skills like navigation or manipulation. As usual, such a layer is implemented using the Robot Operating System (ROS) [20]. Besides means of structuring data communication (topics) and computation (nodes), it provides concepts for implementing behaviors (skills) via the action concept. It allows running a concurrent behavior such as navigating to a goal [18] while reporting intermediate feedback on the progress as well as the final result of the skill execution. Standard implementations such as the ROS navigation stack contain hand-coded monitors (e.g. tracking the progress of the robot pose) but do not provide detailed explanations in the case of a failure. Diagnosers such as the one proposed in [10] that can provide detailed explanations about a navigation fault need to be integrated. Moreover, only rudimentary recovery behaviors are implemented in the navigation stack. Thus, a diagnosis and an interpretation utilizing the corresponding context need to be done at the higher layers. Intermediate observations might be provided by dedicated perception modules such as a localization module that tracks the robot pose [23]. The context and observations of the skill layer is built up by continuous values representing circumstances such as the robot pose ($\langle x, y, \theta \rangle$). Also, more detailed information such as the particle set used in the localization can be utilized to create a local monitor for problems in the navigation [9].

3.2 Executive Layer (Mid-Level)

The task of the executive layer is to refine the abstract actions dispatched by the deliberative layer and supervise the execution of the refined actions. While it is common to use the concept of hierarchical task networks (HTNs) [19] or BDI architectures [14] for the implementation we use the concept of Behavior Trees [8] because of its simpler use and ROS integration. In contrast to a declarative approach such as planning BTs are composed of different nodes (e.g. conditions, sequence, fallback, atomic action) and represent a control policy. The execution leads to a trace of atomic actions that is dependent on the actual development of the context. Any legal termination of a tree counts as a success. If any node in the tree fails a fail is reported. The issue is that BTs can replicate a complete imperative programming language, necessitating runtime verification and full operational semantics for complete diagnosis [22, 3]. For now, we consider a BT as an ordered sequence of sub-tasks (skills) for the sake of simplicity. After the execution of each sub-task, feedback from the low-level is returned, stating the successful or failed execution of the sub-task. In case of failure, the following sub-tasks are not concluded, and the Mid-Level notifies the failed execution of the PDDL action to the High-Level. In case of a successful execution of the entire Behavior Tree, the Knowledge Base is updated at the High-Level with the effect of the PDDL action. However, determining which effects have to be applied and which do not in case of a failure in the execution of the BT is not trivial. In order to make the KB consistent, we need to know the nature of the failure and how it affects the other elements of the environment. For this reason, a local diagnosis needs to be performed on the Behavior Tree, supported by observations made at the Mid-Level and the structure of the entire BT execution. Moreover, such a technique is applied even in the case of positive feedback from the Low-Level. Some failures may not have been detected at the Low-Level because of its limited context. The application of a diagnosis technique, which takes into consideration a temporally broader set of observations (obtained by a sequence of skills rather than one skill in isolation), may help to detect hidden faults in the execution.

3.3 Deliberative Layer (High-Level)

As described above a High-Level PDDL action is refined at the Mid-Level into a Behavior Tree. The diagnoser on the high level is based on History-Based Diagnosis [16] and follows the idea presented in [13]. It is triggered either by the monitor at its level or by failure reports and diagnoses reported from the lower levels. The monitor is currently a simple consistency check between the Knowledge Base and reported observations. As pointed out above the Knowledge Base is updated with an action's effects in the case of its successful execution or by the integration of observations from the lower levels. In the latter case we follow the concept of sensing actions [21] that can update facts in the Knowledge Base besides integrating action effects only. Once an inconsistency is triggered the diagnoser aims to find an alternative sequence of the actions executed so far whose effects are consistent with the Knowledge Base and the recently made inconsistent (unexpected) observation. Alternatives are generated by exchanging actions with a variant modeling a specific fault mode or by adding actions that represent exogenous events. This approach is computationally demanding and requires explicit models for all action faults and exogenous events. If a consistent execution trace is found, the Knowledge Base is updated according to the effects of its actions.

4 What is missing?

The integrated architecture proposed in this paper is still a work in progress. However, the control architecture and monitoring and diagnosis schema have been implemented as a proof of concept and successfully applied in domains like the RoboCup Logistics League (RCLL) [4]. The limitation is that the approach was implemented ad-hoc, with mostly separate concepts at individual levels, and a fully formalized hierarchical diagnosis integration is still lacking. In fact, in the RCLL domain, our diagnostic approach ensures robust and solid execution, preventing from ending up in unresolvable situations and always allowing for some recovery strategy. However, the lack of better integration of diagnostics across different levels and the uncertainty about the true state of the robot and its environment necessitates aggressive recovery strategies, potentially discarding more beneficial situations unnecessarily. The difficulty is here still the proper mapping of the individual representations of contexts, decision-making approaches, execution semantics, diagnosers, and monitors a unified representation. A promising way is shown in [2] where the Situation Calculus and the agent language ConGolog are used as a unified representation across levels in agent supervision. Unfortunately, such methods rely heavily on first-order reasoning and are therefore computationally very expensive. Moreover, also challenges remain on the individual levels such as the complete monitoring and diagnosis of complex Behavior Trees because of the operational semantics. Another important aspect is the reuse of domain knowledge and common sense knowledge which are of utmost importance for feasible monitoring and diagnosis of robot systems interacting with a real open environment. Here still a lot of manual work is needed hindering the automation of knowledge reuse and also limiting the amount of usable knowledge. This is currently particularly interesting as we see Large Language Models (LLMs) as compiled common sense knowledge that is richer and easier to access as attempts like Cyc [17]. A learning module is needed for a complete cognitive architecture to handle novel fault situations and update fault and background models accordingly. While autonomous agents are the goal, interaction with humans remains relevant. The ability of a system to recognize when a situation is unclear and seek human advice would be a significant step toward creating truly dependable agents.

With this paper, we like to stimulate a discussion about the idea of an integrated monitoring and diagnosis approach for cognitive architecture and look forward to valuable feedback on our ideas.

References

1 David W. Aha. Goal Reasoning: Foundations, Emerging Applications, and Prospects. *AI Magazine*, 39(2):3–24, June 2018. doi:10.1609/AIMAG.V39I2.2800.

2 Bita Banihashemi, Giuseppe De Giacomo, Yves Lespérance, et al. Hierarchical agent supervision. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 1432–1440. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2018.

3 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. *Lectures on Runtime Verification: Introductory and Advanced Topics*, pages 1–33, 2018. doi:10.1007/978-3-319-75632-5_1.

4 David Beikircher, Marco De Bortoli, Leo Fürbaß, Thomas Kernbauer, Peter Kohout, Dominik Lampel, Anna Masiero, Stefan Moser, Martin Nagele, and Gerald Steinbauer-Wagner. Robust Integration of Planning, Execution, Recovery and Testing to Win the RoboCup Logistics League. In Cédric Buche, Alessandra Rossi, Marco Simões, and Ubbo Visser, editors, *RoboCup 2023: Robot World Cup XXVI*, pages 362–373, Cham, 2024. Springer Nature Switzerland.

5 Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, 2015.

6 Luca Chittaro, Giovanni Guida, Carlo Tasso, and Elio Toppano. Functional and teleological knowledge in the multimodeling approach for reasoning about physical systems: a case study in diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1718–1751, 1993. doi:10.1109/21.257765.

7 Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, Boca Raton, Florida, USA, 2018.

8 Michele Colledanchise and Peter Ögren. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Transactions on Robotics*, 33(2):372–389, April 2017. doi:10.1109/TRO.2016.2633567.

9 Matthias Eder, Michael Reip, and Gerald Steinbauer. Creating a robot localization monitor using particle filter and machine learning approaches. *Applied Intelligence*, 52(6):6955–6969, 2022. doi:10.1007/S10489-020-02157-6.

10 Matthias Eder and Gerald Steinbauer-Wagner. A fast method for explanations of failures in optimization-based robot motion planning. In Andreas Müller and Mathias Brandstötter, editors, *Advances in Service and Industrial Robotics*, pages 114–121, Cham, 2022. Springer International Publishing.

11 Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, December 2003. doi:10.1613/JAIR.1129.

12 Erann Gat, R. Peter Bonnasso, Robin Murphy, and Aaai Press. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1997.

13 Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer, and Alexander Ferrein. Belief management for high-level robot programs. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

14 F.F. Ingrand, R. Chatila, R. Alami, and F. Robert. Prs: a high level supervision and control language for autonomous mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 43–49, 1996.

15 Félix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 247:10–44, 2017. doi:10.1016/J.ARTINT.2014.11.003.

16 Gero Iwan. History-based diagnosis templates in the framework of the situation calculus. *AI Communications*, 15(1):31–45, 2002. URL: http://content.iospress.com/articles/ai-communications/aic252.

17 Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995. doi:10.1145/219717.219745.

18 Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*, 2010.

19 Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003. doi:10.1613/JAIR.1141.

20 Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

21 Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

22 Matteo Tadiello and Elena Troubitsyna. Verifying Safety of Behaviour Trees in Event-B. *Electronic Proceedings in Theoretical Computer Science*, 371:139–155, September 2022. doi:10.4204/EPTCS.371.10.

23 S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128(1-2):99–141, 2000. doi:10.1016/S0004-3702(01)00069-8.

A Model-Based Approach for Monitoring and Diagnosing Digital Twin Discrepancies

Elaheh Hosseinkhani
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Martin Leucker
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Martin Sachenbacher[1]
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Hendrik Streichhahn
Dräger Safety AG & Co. KGaA., Lübeck, Germany

Lars B. Vosteen[1]
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Abstract

Recent decades have seen the increasing use of Digital Twins (DTs) – that is, digital models used over the lifetime of a physical product or system for tasks such as predictive maintenance or optimization – in a number of domains such as buildings, manufacturing, or design. DTs face a challenge known as the *DT synchronization problem*; a DT, often based on machine-learned, or complex simulation models, needs to adequately mirror the physical product or system at all times, as any deviations might affect the quality of predictions or control actions. In this paper, we present a model-based approach that aims to add a level of awareness to DT models by supervising if they are in sync with the physical counterpart. The approach is agnostic to the type of models used in the DT, as long as they are compositional, and based on monitoring critical properties (behavioral or functional aspects) of the system at run-time. In the case violations are detected, it reasons on the DT's structure to localize and identify parts of the model that cause deviations and need to be adapted. We give a formal description and an implementation of this approach, and illustrate it with an example from building climatisation.

2012 ACM Subject Classification Computer systems organization → Reliability

Keywords and phrases Digital Twins, Runtime Verification, Diagnosis, FDIR, TeSSLa

Digital Object Identifier 10.4230/OASIcs.DX.2024.2

Supplementary Material *Software (Source Code)*: `https://github.com/vosteen/FDIR` [36]
archived at `swh:1:dir:1972d8d2373f5d9c6e127fa16aec1a79755b056a`

Funding This paper is partially funded by the German Federal Ministry for Economic Affairs and Climate Action, due to a resolution of the German Bundestag, under grant number 01MC22005C.

Acknowledgements We want to thank Jan-Henrik Metsch for insightful discussions and valuable feedback.

1 Introduction

In recent decades, steady progress in areas like the Internet of Things, Cyber-physical systems, Model-based engineering and Artificial Intelligence, and Cloud computing has enabled the digitalization of different assets such as products, systems, and processes in

[1] corresponding authors

© Elaheh Hosseinkhani, Martin Leucker, Martin Sachenbacher, Hendrik Streichhahn, and Lars B. Vosteen;
licensed under Creative Commons License CC-BY 4.0

different industrial sectors. Specifically, Digital Twins (DT) have become a popular concept in various application fields such as urban construction, manufacturing, automotive systems, or individualized healthcare [28, 29, 37, 26]. A DT is a set of adaptive models that emulate the behaviour of a physical asset (termed Physical Twin, or PT) in a virtual system, getting real time data to update itself along its life cycle. The DT replicates its physical counterpart to enable and analytically improve tasks like what-if analysis, monitoring and predictive maintenance, or optimisation and parameter tuning based on the communicated data. The practical relevance is underlined by the existence of various (commercial) DT tools and frameworks such as Microsoft Azure's Digital Twins and its corresponding Digital Twins Description Language (DTDL) [22], Plattform Industrie 4.0's Asset Administration Shell (AAS) [1], and INTO-CPS Digital Twin as a Service [4]. Despite this, DTs have notoriously evaded a more standardized or formally strict definition beyond the basic understanding that there exists a digital representation that is kept synchronized at a specified frequency and fidelity with its physical counterpart through communication channels (for instance, see the set of different definitions resulting from a recent Dagstuhl seminar on the topic [10]). This may be attributed to the multi-disciplinary nature of the concept and to the quite broad understanding of digital representations (models), which might range from light-weight visualizations to logical and ontological representations, to machine-learned models and complex numerical simulation models.

Shared by all instances of DT technology, however, is the key challenge that the DT's model needs to adequately mirror the physical product or system, as any deviation might deteriorate the quality of the obtained predictions, alerts, or actions. This problem is often referred to as *DT synchronization problem* ([33, 11]). We use the same term here, but want to note that while the literature emphasizes the role of the communication link between the DT and the PT, we view synchronization more broadly as the problem that the DT needs to faithfully represent the behavior of the PT. As the DT is meant to match the PT along different stages of its entire life cycle, synchronization is a continuous effort and there are potentially several reasons which might cause the DT to be out of sync. They could be roughly classified into failures (component breakdowns) in the PT, parameter drifts e.g. due to wear in the PT, or communication faults between the DT and the PT.

Although various solutions have to date been proposed to the DT synchronization problem, they are typically limited to specific contexts. For instance, Kamburjan et al. [16] propose a solution that exploits ontology models (OWL) and domain-specific constraints to dynamically reconfigure simulators in a DT. Tan and Matta [33] give an overview of different types of DT syncronization problems occuring in the context of simulation and a solution when to best update simulation models based on a cost model.

Of course, the fields of Model-based Diagnosis (MBD) and Fault Detection, Identification and Reconfiguration (FDIR) offer a rich and long-standing body of research about how to derive and maintain model representations that truthfully represent a physical artifact's state, referred to as mode estimation or state estimation, respectively (see e.g. [12, 24, 34]).

However, these approaches are based on specific classes of models (structural or logical models, discrete-event automata, residual equations, etc.), as not all forms of models (e.g., machine-learned or simulation models) are directly amenable to diagnostic reasoning, which requires tracing the system backwards from observed symptoms to inner component states. Actually, in meaningful DT applications the models can be quite large (e.g., city-scale) and heterogeneous (e.g., composed of different simulation units requiring precise timing and orchestration), making first-principles diagnostic and causal reasoning difficult. There also exist methods aiming to transform some types of simulation models (such as Modelica

models) into diagnostic models [23, 27, 18], which could work to address some forms of DT synchronization issues. Also more recently, MBD methods have been used to update/repair specific forms of learned models in the face of additional data [2].

In this paper, we present a different model-based approach to the problem of DT synchronization and fidelity during runtime. Our approach is agnostic to the specific type of DT models, using only structural information (assuming a compositional DT) to be as general as possible. To capture behavioral and functional aspects of the DT that are critical to synchronization, we instead rely on a set of specifications expressed formally in some temporal logic. Our approach then consists of monitoring these properties during run-time using techniques from Runtime Verification (RV). Once violations of properties are detected, MBD on the structural model is used to identify root causes of violations. An approach for coupling RV and MBD has already been presented in previous work by Bauer, Leucker and Schallhart [6]. It is based on structural models and focuses on fault detection and localization in distributed systems as part of a runtime reflection framework. We build on this approach, and adapt it to the context of DTs. Also, we identify several theoretical and practical challenges that arise when coupling RV and MBD in DTs. Using RV for discovering divergences between DTs and their real-world counterpart has also recently been suggested in [13]. However, this work focuses more on conceptual variants and challenges for RV, such as proper specification languages to describe meaningful divergences. In contrast, we present an actual implementation of coupling RV and MBD for DT, and illustrate it with a concrete example.

The remainder of the paper is structured as follows. In the next section, we review relevant background on DTs and RV using monitors. We then present our approach of coupling RV and MBD to identify and localize discrepancies between a DT and its PT, in order to keep the DT in sync. Our prototypic implementation is based on DTDL+, an extension of Azure's DTDL as an executable DT specification, and the temporal stream-based specification language TeSSLa [15] for RV and monitoring. We present experimental results for a small example of a DT for building climatization.

2 Background

2.1 Digital Twins

Digital Twins (DTs) are virtual models that represent real-world entities and systems. They are commonly used for predictive purposes, allowing the simulation of various potential outcomes of the physical counterpart. For a DT to be effective, it must accurately reflect the behavior of the real-world system it represents. Any significant or persistent divergence between the DT and its physical counterpart can result in flawed predictions, incorrect diagnoses, and a distorted understanding of the system's operation. Therefore, discrepancies between the DT and the real-world entity must be continuously monitored and promptly addressed in real time.

Figure 1 illustrates the structure of a DT. The physical entity, referred to as the PT, receives inputs (I), which may be either controllable (user-defined) or uncontrollable (environmental) factors. The PT responds to these inputs by producing outputs (O_p), which could either be actions performed by the PT or measurements of its state or environment. Simultaneously, the inputs are also provided to the DT, which simulates the behavior of the PT and generates its own outputs (O_d). As shown by the dotted lines in Fig. 1, certain outputs from the DT (O_d) may be used to control the PT, while some outputs from the PT (O_p) are fed back to the DT as observational data.

Although sometimes (e.g. Kritzinger et al. [17]) different types of DTs – namely, digital models, digital shadows, and DTs – are further distinguished based on the direction of communication between the PT and DT (one-way or two-way), for our purposes we do not distinguish between these variations.

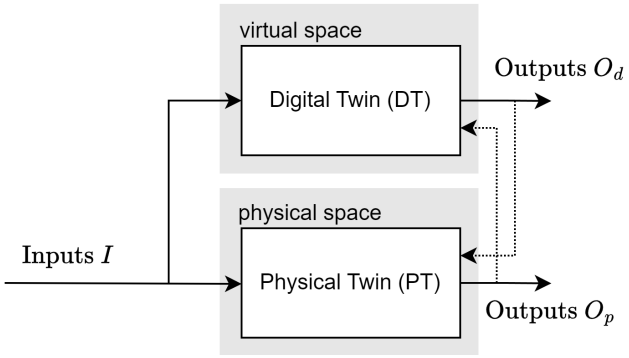


Figure 1 Schematics of a Digital Twin.

In [19], we identified three different artifacts as the key ingredients of a DT's formal representation. These artifacts are

- *environmental models*, such as building models, that provide background knowledge about the structure (e.g. floor plan) and semantics (e.g. accessible doors and stairs in a building) of the DT's context, to represent e.g. locations of people and plan e.g. feasible routes;
- *communication and discrete optimisation*, such as updating the people's positions and finding the shortest path in a building and travelling salesman to search all rooms;
- *physical process models*, that allow to simulate e.g. how long the air supply of a firefighter will last along a route, or the thermal dynamics of heat dispersion in a building.

Like in other fields of model-based engineering, digital twinning incorporates the idea that a DT should be compositional, i.e. composed of (reusable) blocks corresponding e.g. to simulation units or FMUs. For example, in DTDL [22], any DT instance consists of the classes Interface, Component, Command, Property, Relationship, and Telemetry. We use the compositionality of DTs in our approach by exploiting the structure of DTs to reason about possible causes of discrepancies.

2.2 Runtime Verification and Monitoring

RV [20, 5] is a formal method from the area of software engineering. In RV, objects called monitors are used to observe the behavior of programs. The data stream of interest is transmitted to the monitor, producing a sequence of events called traces. A monitor compares this event trace against a formal specification φ that holds in a correct execution. The monitor can report violation or validation of the desired specification. Its verdict is reported on-the-fly, during the execution of the monitored program. Other than testing, RV is a passive procedure that does not actively generate inputs to drive the system. Compared to model checking, which is concerned with mathematically proving properties that must hold in all execution of the system, RV is more lightweight and only concerned with concrete traces of the running system.

Research in RV has produced a variety of monitors that can be automatically generated from specifications φ formulated in variants of linear time temporal logic (LTL), and corresponding higher-level specification languages such as LOLA [3] or TeSSLa [15]. However,

as a monitor can have only a finite view (trace) of the system's behavior, whereas LTL is defined over infinite traces, a semantics has to be defined for RV on finite traces. In [7], a three-valued semantics is proposed for this case, where the monitors can either report $true$, $false$, or ?, the latter meaning an inconclusive verdict, indicating that the trace observed so far does not allow to decide whether φ holds or whether it will be violated in the future.

2.3 Property-based Monitoring of Discrepancies

To discover possible divergences between the DT and its physical counterpart, we have to compare their behavior. One way, as also suggested in [13], is to use conditions $\varphi_1, \ldots, \varphi_n$ which must hold for all executions of the DT. These properties, acting as a form of guarantee, might either be given as a formal specification, deductively extracted from a white-box twin beforehand, or generated by observing multiple executions of the (black-box) twin and learning a specification of its behavior. In any case, we suppose the existence of a set of such conditions, $\varphi_1, \ldots, \varphi_n$, beforehand. The properties can be converted into monitors that then observe the operation of the real-world entity in real time. Any observed sequence that violates one of the properties φ_i is a sequence that cannot be produced by the DT for the same inputs, and thus indicates a divergence between the DT and its real-world counterpart.

2.4 Diagnostic Reasoning from Property Violations

MBD and FDIR deal with the problem of detecting if a system is not functioning correctly, and determining which part of the system is failing (and which kind of fault is present). It is based on reasoning from observations, which provide information on the actual system's behaviour, and in particular discrepancies between values predicted by a model and these actual observations. These discrepancies (called symptoms) are the starting point for diagnosis and the goal is to explain them by hypothesizing (a minimal number of) faults in the system that cause the deviation between predicted and actual behavior.

If properties of the system are monitored, as outlined in the sections above, their fulfillment can be used in lieu of the observations. They can be viewed as adding an additional layer of abstraction: instead of considering discrepancies of values in the system, the starting point for diagnosis are violations of properties by the system. The "classic" notion of symptoms, i.e. discrepancies between predicted and observed values, can be re-gained in this case as a special case of (violated) properties that demand equality of two variables (predicted versus actual). In this sense, monitoring of properties is a powerful concept that allows to capture complex symptoms, involving (temporal) behavior of variables.

However, for diagnosis, this creates the challenge how such generalized "symptoms" – violation of one or more properties on the system's trace – could be attributed to parts of the system failing, and the type of fault(s) occuring. It is not obvious how faults of system components (whose behavior is modelled as propositional logic sentences, finite state machines, etc.) could be related to violation of (arbitrary) temporal logic properties.

To date, a number of approaches have been suggested for coupling run-time property monitoring and MBD. [6] use an approach where the compositional models abstract away from the component's behavior. More precisely, the component models state that if all the inputs of a non-abnormal component are correct, then its output must also be correct, allowing one to infer that it cannot be responsible for violating a property monitored downstream at its output. Though a coarse abstraction, such "behavior-free" models can be used to compute conflicts and diagnoses, and they can be easily derived from the graph structure of the model. Diagnostic models based on propagating deviations [32, 31], where symptoms

correspond to values being either too low, too high or ok, can also be viewed as a form of diagnostic reasoning with property violations, although the properties themselves are not stated explicitly. In [25], the authors present an approach for diagnosis of multi-agent systems where the agents and a system goal are observed across multiple executions (runs), and a spectrum-based fault fault localization (SFL) method is used to explain failed runs by diagnosing which agents are faulty. As it defines the success and failure of an execution based on a property of the system rather than states of the components, this can also been seen as a form of property-based monitoring approach that we aim to formalize in this paper. However, while [25] assumes the evaluation of runs at the very last time step after all executions are finished, our approach uses RV as a fault detection mechanism that can also work on-line.

In the field of RV, assumption-based RV has been proposed in [9] as a framework that leverages assumptions on the behaviors of the system under scrutiny for reasoning on its non-observable (internal) or future behaviors. The idea is that the monitor outputs (property satisfied or not satisfied) are conditioned on the assumption that the involved parts of the system are behaving correctly, allowing one to reason about these assumptions in a model-checking framework.

3 Building Climatization Example

In an on-going project[1], novel approaches for designing, deploying, and monitoring DTs are developed in the context of buildings and emergency scenarios. Specifically, to enhance the DT model's accuracy and longevity, the aim is to add awareness regarding some features and to supervise the DT's performance through monitoring and fault localization. Loosely derived from this project, we present a simplified example to motivate and illustrate our approach[2]. The example system consists of two parts, the PT and the DT:

Physical Twin. Consider a makeshift building (tent) separated into three heating zones. Each zone has a heater (H_1, H_2, and H_3), controlled with an on-off power value $h_i \in \{0, 1\}$. The temperature within the zones is measured by four sensors, T_A to T_D, strategically placed at the periphery and between the heating zones. Each sensor provides a measurement t_i in °C. Figure 2 provides a schematic overview.

Digital Twin. The primary task of the DT is to maintain the temperature in each zone above 20 °C to ensure occupant comfort, while optimising for energy efficiency by not heating unnecessarily. Each zone has its own heating controller (C_1 to C_3), each of which sends a control signal. To do this, the temperature propagation in the PT is simulated within the controller to inform the decisions. In the problem classification mentioned in 2.1, this represents a physical simulation and is approximated by an ordinary differential equation that is solved iteratively. The rest of the DT system is based on digital processes.

The second purpose of the DT is to increase reliability. To this end, the monitors M_A to M_D monitor the output values of the temperature sensors T_A to T_D and identify zones where the temperature targets are not being met (significantly below 20 °C) or are not progressing satisfactorily (e.g, the temperature t has not increased by 5 % from the initial value t_0 within 42 s). Whenever a monitor is triggered, we want to identify components that may be responsible for the fault. In addition, the heater controllers monitor their internal simulation by keeping track of whether they are providing results that match the measurements from the PT. Whenever a loss of simulation alignment is detected, an automatic synchronisation of the simulation model to better fit reality should be initiated.

[1] `https://o5g-n-iot.de`
[2] The source code of the example is available in the repository `https://github.com/vosteen/FDIR`

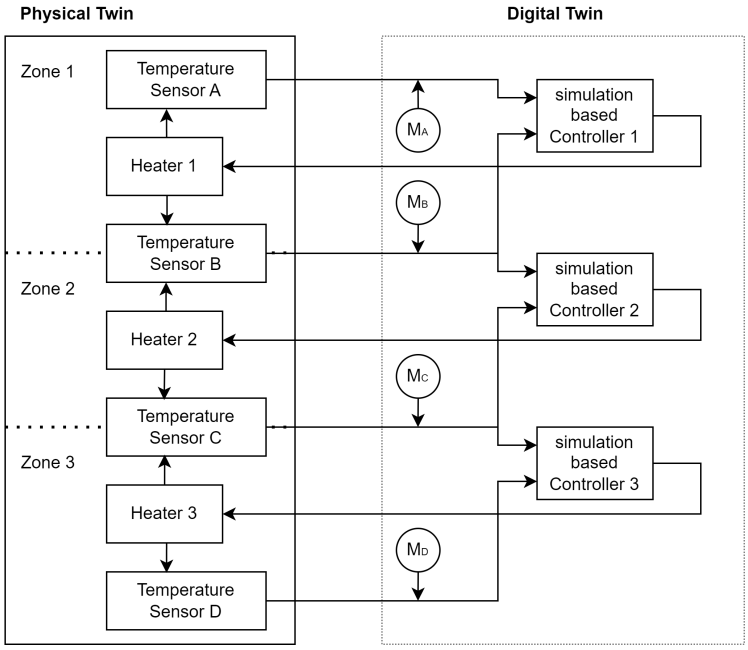


Figure 2 Schematic overview of the running example.

4 General Approach

In this section, we combine the principles laid out in Section 2 to propose an approach for detecting discrepancies between a DT and its PT, i.e. checking the synchronicity of DT, and localizing possible root causes (diagnosis) at runtime.

4.1 A Model for Property-based Diagnostic Reasoning

We consider a system (DT) composed of a finite set of components, denoted $COMP$. Diagnosis aims to determine whether a subset of the components is faulty (leading to discrepancies between DT and PT). A special predicate, AB, is used to denote that a component is abnormal, i.e. behaving different from its intended or specified behavior.

As usual in MBD, the system is then represented as a tuple $S = (SD, COMP)$, where $COMP$ are the components and SD is the system description, comprised e.g. of a set of first-order sentences (the exact type of model is not relevant in the following). We assume the components are directed, i.e. they have inputs and outputs (for simplicity, we assume a single output).

The main difference of our approach to "classical" MBD is in the definition and use of observations, typically denoted OBS. We assume a set of properties $\varphi_1, \ldots, \varphi_n$, denoted $PROP$, which must hold for all executions of the DT. We use the properties – more specifically, the verdicts (outputs) of corresponding monitors constructed from these properties – in lieu of direct observations of the system. This leads to a further abstraction of the system model: instead of observations of actual system values, we only use the information whether the values conform to the specified properties. Since the properties we are after are in general defined on streams of values instead of single values, we refer to "channels" instead of values in our model. Following the same ideas as in [6], we use a predicate *ok* to denote whether a channel behaves according to the specified properties, and devise a model that only captures the causal dependencies between the channels.

The system description is then reduced to a set of first-order sentences describing the correctness of input-output behavior of the components and channels. That is, for every component $C_j \in COMPS$ with inputs $i_1, \ldots, i_m$ and output o we get a formula

$$ok(i_1) \wedge ok(i_2) \wedge \cdots \wedge ok(i_m) \wedge \neg AB(C_j) \implies ok(o)$$

where ok denotes that a value or channel does not violate any specified property, and $\neg AB$ denotes that the component is not abnormal. As described in Section 2.2, the monitors check whether a sequence of events satisfies a certain property $\varphi_k \in PROP$. The monitors can then report either $true$, $false$, or inconclusive (denoted ?). For diagnosis we are interested in finding the causes of violations of properties, and we can thus identify $true$ and ? [6]. Thus, as observations OBS of the system, we get the information whether certain values (channels) in the system are ok (if the monitor reports $true$ or ?) or $\neg ok$ (if the monitor reports $false$). More formally, let v_k denote the values (channels) monitored by the properties φ_k. Then the observations are

$$OBS = \{ok(v_k) \mid \varphi_k = true \vee ?\} \cup \{\neg ok(v_k) \mid \varphi_k = false\}$$

In sum, this gives us the familiar notion of consistency-diagnosis, but with the specific interpretation of SD and OBS outlined above.

4.2 Extending the Model to Temporal Property-based Diagnostic Reasoning

The model described in the section above, taken from [6], abstracts away from the components internal behavior, in particular also their temporal behavior. It assumes that the *ok*-predicates for the outputs are generated instantaneously from the *ok*-predicates of the inputs of the components.

Considering the different artifacts of a DT as outlined in Section 2.1, this assumption appears reasonable for the "digital", i.e. the *communication and discrete optimisation* components of a DT. In fact, if we apply the modeling approach described in Section 4.1 and [6] to the DT scenario from Section 3, including the *physical process models*, a problem appears: as the example contains a physical process (heat dissipation) together with sensing, control and actuating components, a causal loop will be created in the model reasoning about the *ok*-predicates. The consequence (for the example, but also in general) is that the diagnostic model would in this case yield all components as possible diagnoses, not allowing any further discrimination.

Looking more closely, the underlying problem is that the model from Section 4.1 is a too coarse abstraction of the DT; we must at least distinguish the *communication and discrete optimisation* models from the *physical process models*. In this example, switching on a heater component has an influence on the zone's temperature, but due to thermal inertia, the effect will be delayed and manifest only in a later time step. The interaction between the heater components and the temperature sensors is thus via the thermodynamics of the respective zone, which could be governed by differential equations and in any case happens at a different time scale compared e.g. to the interaction channel between the sensors or controllers and the controllers and the heaters. Only if we model these interactions with underlying physical processes in a more fine-grained way, we can relate the observations from the monitors in the control loop in a way that is meaningful for diagnosis.

In the following, we present a simple extension to the model from Section 4.1 and [6] to adequately deal also with *physical process models* that are present in a DT. For this, we augment the values (channels) v_i in the model with (discrete) time steps $t = t_0, t_1, \ldots$ such

that values $v_i^{t_0}, v_i^{t_1}, \ldots$ at different time steps can be distinguished. We are not concerned here with how such time-discrete models are obtained, but note that there are methods to generate them from underlying differential equation models by abstraction, as e.g. in [21].

Using the extension of values to time steps v_i^t, we can extend the causal-structural model from Section 4.1 to include also components that relate variables (in particular, the *ok*-predicates of variables) at different time steps (for simplicity, we limit ourselves to consecutive time steps):

$$ok(i_1^t) \wedge ok(i_2^t) \wedge \cdots \wedge ok(i_m^t) \wedge \neg AB(C_j) \implies ok(o^{t+1})$$

For the example, the interaction between the heaters and the temperature sensors (via a physical process) can be captured with such time-augmented formulas, whereas the other components in the example can be modeled as in Section 4.1. Together, we get a diagnostic model of both the DT's *physical process models* and *communication and discrete optimisation* components that describes the causal input-output propagation behavior. In Section 5, we describe how this model and the observations from the monitors can be used to compute diagnoses.

4.3 A Formal Digital Twin Description Language

Our diagnostic approach requires two pieces of formal knowledge for a DT: The system description SD with components $COMPS$ and their connecting channels, and the specification of properties $PROP = \varphi_1, \ldots, \varphi_n$ which are then turned into monitors to yield the observations OBS. To represent this information about the DT, we developed an extension of the existing DTDL, which we call DTDL+ [30]. While DTDL describes only abstract classes (interfaces) of DT components and their relationships, DTDL+ enriches it with an `Instance` class. This allows to describe concrete instances of DTs or DT components (for instance, several instances of the same component type), as required to specify system descriptions SD. The `Instance` class includes also complete deployment information, including package installation, file transfer, service creation, and the type of data transfer to be used in the DT. DTDL+ descriptions can be automatically translated into various specialised forms such as Ansible playbooks, which enable deployment and reconfiguration. The current description focuses primarily on the DT components that drive the use case.

In addition, there are already concrete ideas for the automated integration of further essential elements such as:

- the reality or its simulation,
- the visualisation component,
- the monitors, and
- the identifiers, which can then automatically utilise the DTDL+ description.

The integration of these aspects is currently in progress.

5 Implementation and Application to the Example

We have implemented the approach using DTDL+ for the DT model specification, TeSSLa for generating runtime monitors from specifications, a numerical simulation framework, and a SAT solver to compute diagnoses as described in Section 4. The resulting setup for the example is shown in Fig. 3. It consists of a set of components designed to simulate, control and monitor the thermal environment of a building structure. The source code can be found in our repository[3].

[3] `https://github.com/vosteen/FDIR`

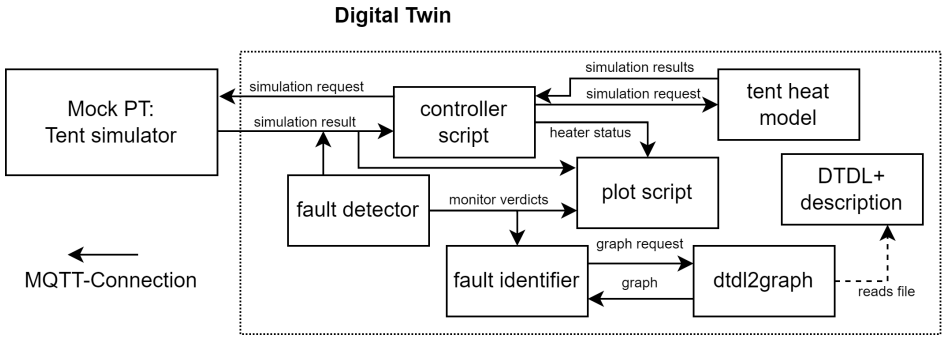


Figure 3 Experimental setup of the implementation used for the example.

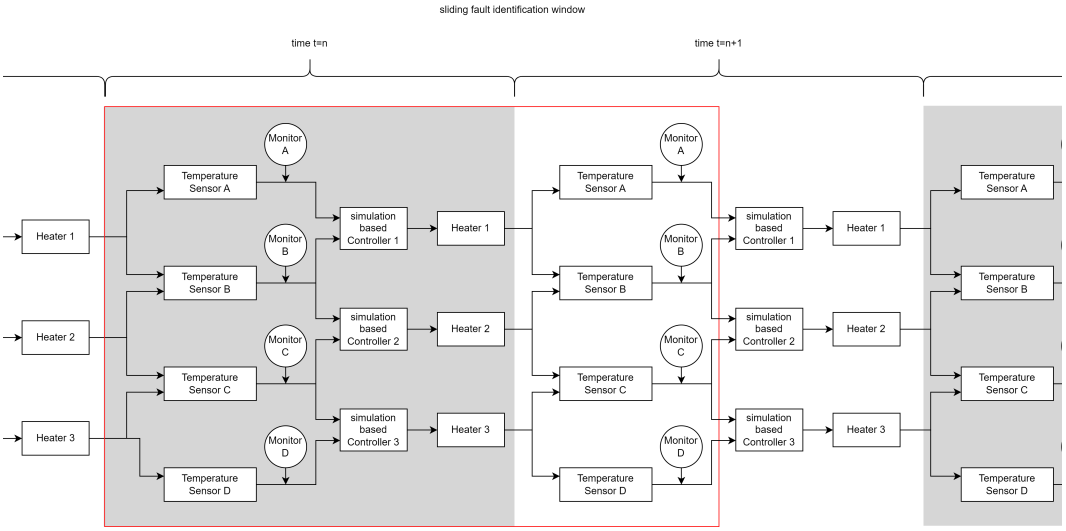


Figure 4 Diagnostic model of the DT unrolled for a horizon of two time points.The red box presents the subgraph that is used for fault identification.

At the core of the system is the Mock PT, acting as a thermal simulator that processes heating control signals and provides temperature projections for the next time step. This simulation is based on the ISO 52016-1 standard [14], adapted from [35]. The heater controller determines whether to activate a heater based on the digital twin's internal simulation. If the simulated temperature for the controlled zone is predicted to drop below 20 °C, the heater is switched on or remains on for the next cycle. The Mock PT is used to evaluate the fault diagnosis system. The digital twin can be automatically deployed using DTDL+ descriptions. The controller regulates the tent's temperature by switching the heaters. Heater statuses and sensor data are displayed graphically via plot scripts, while DTDL+ descriptions are processed to generate a fault identification graph. Faults detected by the monitoring systems are identified using a SAT-based model. For fault detection, we use a monitor to listen to temperature sensor data streams (see Fig. 2 for an overview). These streams report to the fault identification process.

Towards identifying faults in components, the monitoring component continuously evaluates the temperature data according to the following rule. In TLTL [8] this condition can be formalised as $G_{[t-42\,\mathrm{s},\,t]}\,(\mathrm{temp} < 19.5\,^{\circ}\mathrm{C}) \wedge (\mathrm{temp}(t) \geq 1{,}05 \times \mathrm{temp}\,(t - 42\,\mathrm{s}))$, and is used to generate the respective four monitors M_a to M_D. This TLTL condition specifies two possible scenarios: First, that the temperature in the last 42 s ago was greater than 19.5 °C. Second,

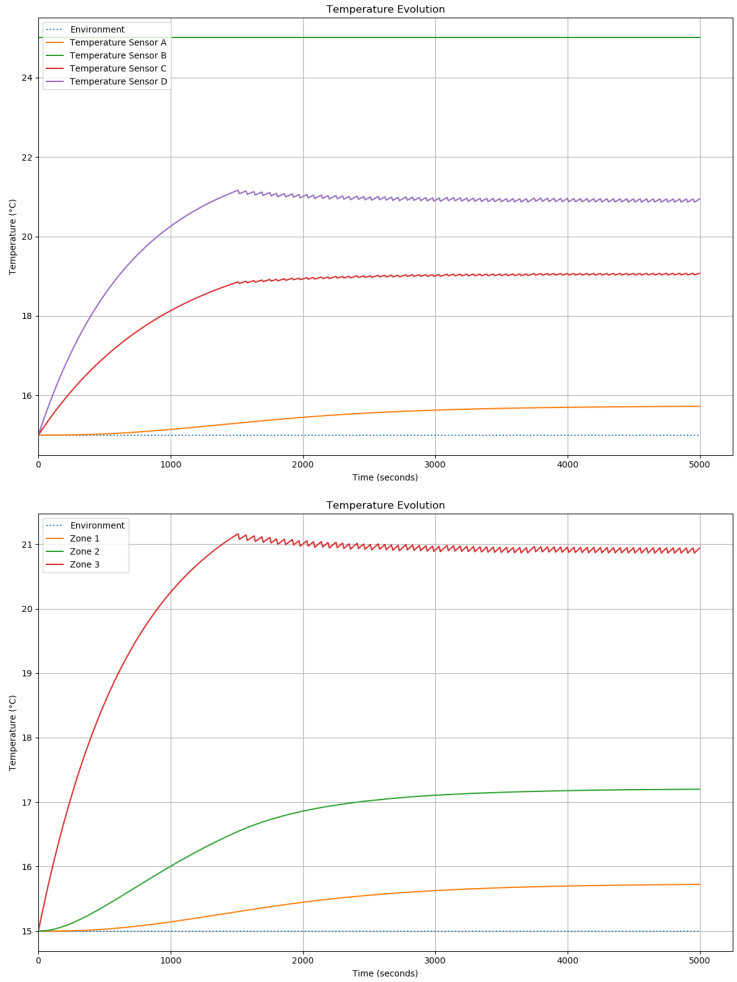


Figure 5 The plot for the temperature sensor on the top shows the faulty T_B stuck at 25 °C. The heater controllers of Zone 1 and 2 average the temperature signals from T_B and T_A/T_C and see their objective (20 °C) reached. As seen in the plot of the corresponding simulated zone temperatures on the bottom, it is actually not reached.

that the current temperature has increased by at least 5% compared to the temperature from 42 seconds ago. The condition is satisfied if either one of these scenarios is true. When this monitor detects a violation, it triggers the fault identifier, which then uses a diagnosis to analyse the data and determine the possible causes of these anomalies. We use the SAT-based approach proposed by Bauer et al. [6], transforming monitor verdicts and the diagnostic model into a SAT instance. Since our example involves temporally extended component models, we handle temporally indexed variables by considering each component as a separate entity at each time step. To limit the diagnosis problem to a finite instance, we unroll the model over a fixed time horizon. Fig. 4 shows an example unrolled for two time points, which is sufficient for accurate diagnosis in our case. The time horizon can be optimized for faster diagnosis or completeness. The diagnosis provides sets of components that, if abnormal, would explain the monitor's observations. To get an up-to-date component graph, the diagnosis triggers the `dtdl2graph` script to extract one from the DTDL+ description. This is sufficient to create the system description part of the SAT problem from [6].

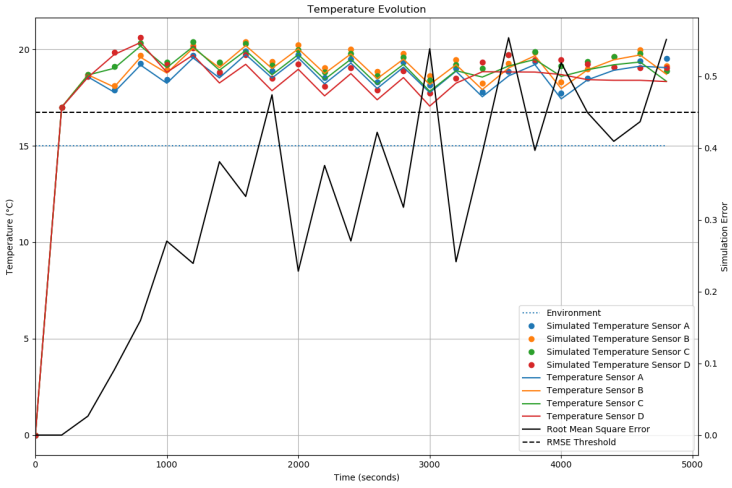


Figure 6 Example of parameter drift and periodic resychronisation: whenever the difference between the PT and the internal simulation of the controller grows too large (as indicated by the black line crossing the limit shown with the dotted line), the simulation is adapted to better predict reality.

To create a diagnostic scenario, a fault can be injected for one or multiple components in the simulation. We take Temperature Sensor B as an example and configure it to provide the constant and (mostly) false temperature signal 25 °C. Figure 5 shows a temperature plot of this simulation run. The error leads to a violation of the monitors M_A and M_C, that is, these temperature values are observed as $\neg ok$. The subset-minimal diagnoses are then determined as follows:

- T_B is abnormal,
- T_C, T_A are both abnormal,
- H_3, T_A are both abnormal,
- ...

The diagnoses could be used as a basis for recovery actions to bring the DT back in sync.

As a scenario for a loss of simulation synchronicity, let's assume the controllers keep the internal simulation results upon which they based their last decisions in memory and compare it to the (simulated) measurements from the PT. If the difference grows too large, the controller-intern monitors detect a fault and trigger a recalibration of the simulation to the measurements. Fig. 6 shows the results of this approach on our example. We then employ a simple gradient descent to find a new set of simulation constants to determine updated (locally) optimal simulation parameters to re-establish synchronicity with the PT.

6 Conclusion

DTs are nowadays a well-established engineering concept for model-based design, development, operations and maintenance for Cyber-physical systems. Thus, besides the development phase, they must also support the evolution of the system and track their physical counterparts (PTs) throughout the whole lifecycle, creating the necessity to update the DT when the physical entity changes.

We presented an approach that uses RV to detect if the DT is *out of sync* with its physical counterpart, and uses MBD to localize the reasons for discrepancies of a DT with its PT in an abstracted model representing the causal structure of the DT. For this, we

extended an approach from [6] to be able to deal also with physical process models that exhibit dynamic behavior. Our approach also includes the development of DTDL+ as a description language that extends Azure's DTDL to executable formal specifications, and the temporal stream-based specification language TeSSLa for RV and monitoring. An example from a building DT and its prototypical implementation illustrates the effectiveness of the approach. We currently work to expand the approach to more complex examples in the area of DT for emergency mission support [19].

The approach is a step in the ongoing development of resilient DTs, that incorporate a level of self-awareness by automatically monitoring their own performance and fidelity and identifying root causes in the case of critical violations.

References

1 Plattform Industrie 4.0. Asset Administration Shell. Accessed on August 19, 2024. URL: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/AAS-ReadingGuide_202201.pdf?__blob=publicationFile&v=1.

2 Shaked Almog and Meir Kalech. Diagnosis for Post Concept Drift Decision Trees Repair. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 23–33, August 2023. doi:10.24963/kr.2023/3.

3 Ben Angelo, Sriram Sankaranarayanan, Cesar Sanchez, and Many Others. Lola: Runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 166–174, July 2005. doi:10.1109/TIME.2005.26.

4 INTO-CPS Association. Digital Twin as a Service. Accessed on August 19, 2024. URL: https://into-cps-association.github.io/DTaaS/.

5 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-75632-5_1.

6 Andreas Bauer, Martin Leucker, and Christian Schallhart. Model-based runtime analysis of distributed reactive systems. In *Proceedings of the Australian Software Engineering Conference (ASWEC'06)*, page 243–252. IEEE, IEEE, 2006.

7 Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Tasiran, editors, *Runtime Verification, 7th International Workshop, RV 2007, Vancouver, Canada, March 13, 2007, Revised Selected Papers*, volume 4839 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2007. doi:10.1007/978-3-540-77395-5_11.

8 Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4), September 2011. doi:10.1145/2000799.2000800.

9 Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification*, pages 165–184, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-32079-9_10.

10 Loek Cleophas, Thomas Godfrey, Djamel Eddine Khelladi, Daniel Lehner, Benoit Combemale, Bernhard Rumpe, and Steffen Zschaler. Model-Driven Engineering of Digital Twins (Dagstuhl Seminar 22362). *Dagstuhl Reports*, 12(9):20–40, 2023. doi:10.4230/DagRep.12.9.20.

11 Digital Twin Consortium. The DTC glossary. Accessed on August 19, 2024. URL: https://www.digitaltwinconsortium.org/glossary/.

12 Johan de Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2003, Washington DC, 9-11 June 1997. doi:10.1016/S1474-6670(17)36467-4.

13 Sylvain Hallé, Chukri Soueidi, and Yliès Falcone. Leveraging runtime verification for the monitoring of digital twins. In Stefan Hallerstede and Eduard Kamburjan, editors, *Proceedings of the Workshop on Applications of Formal Methods and Digital Twins co-located with 25th International Symposium on Formal Methods (FM 2023), Lübeck, Germany, March 06, 2023*, volume 3507 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3507/paper1.pdf.

14 Energy performance of buildings – Energy needs for heating and cooling, internal temperatures and sensible and latent heat loads. Standard, ISO, 2017.

15 Hannes Kallwies, Martin Leucker, Malte Schmitz, Albert Schulz, Daniel Thoma, and Alexander Weiss. TeSSLa – an ecosystem for runtime verification. In Thao Dang and Volker Stolz, editors, *Runtime Verification*, pages 314–324, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-17196-3_20.

16 Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, S. Lizeth Tapia Tarifa, David Cameron, and Einar Broch Johnsen. Digital twin reconfiguration using asset models. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Practice*, pages 71–88, Cham, 2022. Springer Nature Switzerland.

17 Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. doi:10.1016/j.ifacol.2018.08.474.

18 Mattias Krysander, Erik Frisk, Ingela Lind, and YIva Nilsson. Diagnosis analysis of modelica models. *IFAC-PapersOnLine*, 51(24):153–159, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018. doi:10.1016/j.ifacol.2018.09.549.

19 Martin Leucker, Martin Sachenbacher, and Lars Bernd Vosteen. Digital twin for rescue missions - a case study. In Stefan Hallerstede and Eduard Kamburjan, editors, *Proceedings of the Workshop on Applications of Formal Methods and Digital Twins co-located with 25th International Symposium on Formal Methods (FM 2023), Lübeck, Germany, March 06, 2023*, volume 3507 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3507/paper4.pdf.

20 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOSâ'07). doi:10.1016/j.jlap.2008.08.004.

21 Jan Lunze and Jörg Raisch. Discrete models for hybrid systems. In Sebastian Engell, Goran Frehse, and Eckehard Schnieder, editors, *Modelling, Analysis, and Design of Hybrid Systems*, pages 67–80, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

22 Microsoft. Azure Digital Twins. Accessed on August 19, 2024. URL: https://azure.microsoft.com/de-de/products/digital-twins/.

23 Raj Minhas, Johan Kleer, Ion Matei, and Bhaskar Saha. Using fault augmented modelica models for diagnostics. In *10th International Modelica Conference*, 2014. doi:10.3384/ecp14096437.

24 Igor Mozetič. Model-based diagnosis: An overview. In Vladimír Mařík, Olga Štěpánková, and Rorbert Trappl, editors, *Advanced Topics in Artificial Intelligence*, pages 419–430, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

25 Avraham Natan, Meir Kalech, and Roman BartÃ¡k. Diagnosis of intermittent faults in multi-agent systems: An SFL approach. *Artificial Intelligence*, 324:103994, 2023. doi:10.1016/j.artint.2023.103994.

26 Samuel D. Okegbile, Jun Cai, Dusit Niyato, and Changyan Yi. Human digital twin for personalized healthcare: Vision, architecture and future directions. *IEEE Netw.*, 37(2):262–269, 2023. Conference Name: IEEE Network. doi:10.1109/MNET.118.2200071.

27 Bernhard Josef Peischl, Ingo Pill Pill, and Franz Wotawa. Abductive diagnosis based on modelica models. In *27th International Workshop on Principles of Diagnosis*, 2016.

28 Dimitrios Piromalis and Antreas Kantaros. Digital twins in the automotive industry: The road toward physical-digital convergence. *Applied System Innovations Using Recent Communications and Networking Advances*, 5(4):65, 2022. doi:10.3390/asi5040065.

29 Gerhard Schrotter and Christian Hürzeler. The digital twin of the city of zurich for urban planning. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1):99–112, 2020. doi:10.1007/s41064-020-00092-2.

30 Hendrik Streichhahn. Specification of digital twins for a practicable implementation. Master's thesis, Institute for Software and Programming Languages, University of Lübeck, Lübeck, Germany, October 2023. Supported by Lars B. Vosteen.

31 Peter Struss. Deviation models revisited. In *Working Papers of the 18th International Workshop on Qualitative Reasoning*, 2004.

32 Peter Struss, Martin Sachenbacher, and Florian Dummert. Diagnosing a dynamic system with (almost) no observations. In *Workshop Notes of the 11th International Workshop on Qualitative Reasoning*, 1997.

33 Baris Tan and Andrea Matta. The digital twin synchronization problem: Framework, formulations, and analysis. *IISE Transactions*, 56(6):652–665, 2024. doi:10.1080/24725854.2023.2253869.

34 Louise Travé-Massuyès and Teresa Escobet. *BRIDGE: Matching Model-Based Diagnosis from FDI and DX Perspectives*, pages 153–175. Springer, June 2019. doi:10.1007/978-3-030-17728-7_7.

35 Tim Troendle. Simple simple: A simple building energy model. https://github.com/timtroendle/simple-simple, 2024. Accessed: 2024-09-04.

36 Lars B. Vosteen. FDIR for Digital Twins. Software, swhId: swh:1:dir:1972d8d2373f5d9c6e127fa16aec1a79755b056a (visited on 2024-11-14). URL: https://github.com/vosteen/FDIR.

37 Guanghui Zhou, Chao Zhang, Zhi Li, Kai Ding, and Chuang Wang. Knowledge-driven digital twin manufacturing cell towards intelligent manufacturing. *International Journal of Production Research*, 58(4):1034–1051, 2020. doi:10.1080/00207543.2019.1607978.

A Review of Fault Diagnosis Techniques Applied to Aircraft Air Data Sensors

Lucas Lima Lopes
Airbus Operations S.A.S, Toulouse, France
LAAS-CNRS, Toulouse, France

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, France

Carine Jauberthie
LAAS-CNRS, Université de Toulouse, UPS, France

Guillaume Alcalay
Airbus Operations S.A.S, Toulouse, France

Abstract

Air data sensors provide essential measurements to ensure the availability of autopilot and to maintain aircraft performance, flight envelope protection and optimal aerodynamic surfaces control laws. The importance of these sensors imply the existence of embedded fault tolerance features, mainly represented by hardware redundancy. The latter is prone to fail in case of common fault of multiple sensors, especially if the faults are coherent and simultaneous. Increasing the robustness of fault detection and isolation (FDI) techniques for air data sensors to the aforementioned conditions is essential for the development of more autonomous aircraft, reducing crew workload and guaranteeing flight protections under adverse conditions. This paper reviews recent works on Air Data System (ADS) FDI, assessing proposed model, data and signal-driven approaches. We finally argue in favor of data-driven and hybrid approaches for the development of virtual sensors and semi-supervised anomaly detectors, offering an overview of ways forward.

2012 ACM Subject Classification Applied computing → Avionics

Keywords and phrases air data, FDI, aeronautics, review, survey, diagnostics, fault

Digital Object Identifier 10.4230/OASIcs.DX.2024.3

Funding This work was carried out as part of a CIFRE PhD project in collaboration between the LAAS-CNRS laboratory and AIRBUS. It is partially supported by 3IA Artificial and Natural Intelligence Toulouse Institute (ANITI), French "Investing for the Future - PIA3" program under the Grant agreement "ANR-19-PI3A-0004"

1 Introduction

Modern aircraft have steadily been equipped with more and more critical capabilities and functionalities for safe flight and to assist pilots on their tasks in an ever more complex environment. The Flight Control and Management System (FCMS) sustains many of these capabilities. It requires a series of reliable measurements provided by different sensor systems to ensure the availability of autopilot and to maintain aircraft performance, flight envelope protection and optimal aerodynamic surfaces control laws. Alongside important sensor systems such as the Inertial Reference System (IRS) and the Global Positioning System (GPS), the aircraft is equipped with the Air Data System (ADS). The ADS provides the aircraft computers with a series of variables calculated from anemometric and clinometric measurements, arguably the most important parameters for aircraft operation.


© Lucas Lima Lopes, Louise Travé-Massuyès, Carine Jauberthie, and Guillaume Alcalay;
licensed under Creative Commons License CC-BY 4.0



As an essential system, many layers of fault tolerance are applied to prevent and remediate ADS sensor and computer faults. Safety is among the biggest concerns in aeronautics since the earliest aircraft developed, and therefore there is no lack of standard practices and methods for ensuring fault tolerance. In general, these practices involve physical installation segregation, redundancy, dissimilarity, reconfiguration and safety analysis [31].

Hardware redundancy is especially important in the case of ADS sensors fault tolerance and the majority voting logic is widely used for the consolidation of the measurements of the redundant air data sensors. Although they have shown to be reliable numerous times throughout the history of aviation, there are still known cases of coherent and simultaneous faults of two or more sensors of one type that can deceive the voting logic. This has lead the development and on-boarding of new algorithmic solutions for more than two decades, complementing hardware redundancy with analytical redundancy.

Many cases of fault affecting multiple sensors simultaneously and consistently (common mode sensor faults) have been identified: icing is an event that takes place when the flight trajectory crosses a zone at or below freezing temperature with enough humidity. This causes supercooled water droplets to freeze upon contact with the aircraft surfaces [29] and it may block or disturb multiple sensors; lightining strikes might damage and even melt multiple probes [2]; structural damages on the nose radome might disturb airflow around multiple sensors in unpredictable ways [9]; maintenance errors are a common source of issues, the most classic being non-removal of Pitot tube covers, making total pressure measurements to remain stuck in an erroneous value. Although these and other sources of common fault modes have been identified, they do not always provoke the same fault signature. Furthermore, there are still potentially non-identified sources of coherent and simultaneous faults. Therefore generalization is a main concern for FDI on air data sensors.

Previous studies have produced the methods and techniques that ensure the safety required by the aeronautical industry. However, the development of more robust and accurate air data sensor fault diagnosis is an enabler of a more autonomous aircraft, reducing pilot workload and avoiding the loss of automatic protections and auto-pilot. This has renewed the interest in the field of air data sensor fault diagnosis and parameter estimations, resulting in a series of studies and new technologies that expand on the extensive safety barriers already in place. The review presented in this article intends initially to survey the recent studies on the field and present methodologies applied on similar applications that could be exploited in air data diagnosis. In section 4, the current limitations of state-of-the-art and what we consider to be promising lines of research are discussed. Finally, in the conclusion, the most important points discussed and ways forward presented are summarized.

2 Air data sensors

The ADS relies on a series of sensors and processing units. Although dependent on aircraft architecture, these sensors are usually of four types:

- **Static pressure** (P_S) ports: they are visible on the fuselage of the aircraft as round openings. Pressure gauges connected through tubes to the port measure the exterior air static pressure . In the A320 family, three redundant P_S measurements are available.
- **Pitot tubes**: total pressure is an essential concept in fluid dynamics originating in the Bernoulli's equations and is the sum of the fluid static pressure and the dynamic pressure, due to movement of the fluid [6]. The Pitot tube is a classic tool designed to estimate the speed of the air flow by measuring the total pressure. In the A320 family, three redundant P_t measurements are available.

- **Angle-of-attack vanes**: the sensors that measure the aerodynamic angles are usually clinometric vanes that align themselves with the airflow around them, measuring its relative angle to the body axis. A series of positional corrections allow to calculate the angle-of-attack of the airflow around the whole aircraft. Three AoA vanes are present on the A320 family.
- **Total air temperature probes**: measures the total air temperature by decelerating the flow adiabatically in a similar manner to the described Pitot tube. The temperature of the halted air is then measured. A320 family aircraft possess 2 TAT probes.

The measurements from these instruments are used to calculate important flight variables, that are then corrected through the use of exogenous information. The most important resulting output used by flight control and autopilot system or to be displayed to the pilot are indicated air speed, CAS, AoA, β, M, dynamic pressure and baro-corrected altitude/vertical speed.

Physical equations relating the different ADS observations and derivative parameters exist. These are however often dependent on external parameters, some directly measured by IRS, GPS or engine system, some not directly measured, such as wind speed, static air temperature (T_s) and aerodynamic coefficients. Surrogate models often exist for the latter [4], but these are not always reliable. Therefore, the classic FDI approaches of working with analytical redundancy relations derived from minimally structured over determined sets is often hard to apply onboard for ADS diagnostic.

Many of the reviewed approaches propose instead the use of observers to deal with model uncertainties and not measurable inner states for use in diagnostic and variable estimation. The kinematic and dynamic equations relating these variables are however non-linear, as shown in section 3.2, model uncertainties are also often not of zero mean and tend to vary throughout flight. Intense gradient winds and atmospheric turbulence are often not well covered by dynamic models, while integration of IRS and GPS measurements throughout time tend to yield prohibitively large errors for reliable diagnostic in many cases [54].

This comes to illustrate the many challenges of diagnosing faults on air data sensors. Alongside that, diagnostic has to be robust as false-alarms can have very negative impacts, taking pilots attention unnecessarily and triggering unnecessary corrective actions.

3 State-of-the-art ADS sensors fault diagnosis

3.1 Overview

As previously mentioned, the objective of this work is to perform as exhaustively as possible the review of the publicly available research works on ADS FDI. Furthermore, it is our interest to classify and group this work into the generally accepted categories of FDI methods and into logical clusters defined from an utility point of view.

Our first defined classification criterion of the reviewed methods was with respect to its source of analytical knowledge, using the generally accepted separation into model-driven, data-driven and signal-driven techniques [38]. Model-driven methods are those that rely on a physical model of the system of interest, this physical model will provide the analytical redundancy necessary for the fault detection and isolation. Data-driven model extract the system knowledge needed for an analytical redundancy interpretation from historical data on the operation of the system, in this case real flight data or simulated data. Signal-driven methods employ time-domain and frequency-domain tests on the signals of the measurements to detect faults based on expert rules, such as admissible signal saturation and nominal power spectrum.

The second defined classification criterion is made with respect to the type of information output of the methods. We defined two categories: virtual sensors (VS) and non-VS anomaly detectors. Virtual sensor (VS) is a term generally used to a system capable of estimating the value of a variable without directly measuring it, therefore using the measurement of other variables and explicit or implicit analytical relations to estimate the variable of interest. With VSs, FDI is performed by comparing the estimated value against the measured value following a defined comparison logic. Non-VS anomaly detectors is the term used in this paper to designate methods that allow for fault detection and isolation without directly estimating the value of the interest variable. VS will generally provide more information about the fault, allowing for its characterization, and often allow for continued operation through the rejection of the measurement and adoption of the estimation. On the other hand, non-VS anomaly detectors will often not allow for fault characterization and should be combined with additional fault tolerance techniques to allow for continued operation, such as hardware redundancy or live maintenance.

Further classification criteria will be discussed in the dedicated sections of this paper.

More than 40 papers describing novel FDI and variable estimation algorithms and techniques applied to air data sensors were evaluated, a map for reader reference is provided in figure 1. It is observed that the majority of these presented model-based techniques, followed by data-driven techniques. Signal techniques, although globally used as a layer of fault diagnosis in many aircraft system [31], do not have strong presence in the academic research prepared in the last 30 years on the subject. Hybrid methods are a minority as well. 80% of the analyzed papers propose VS architectures, the other part proposing non-VS anomaly detectors.

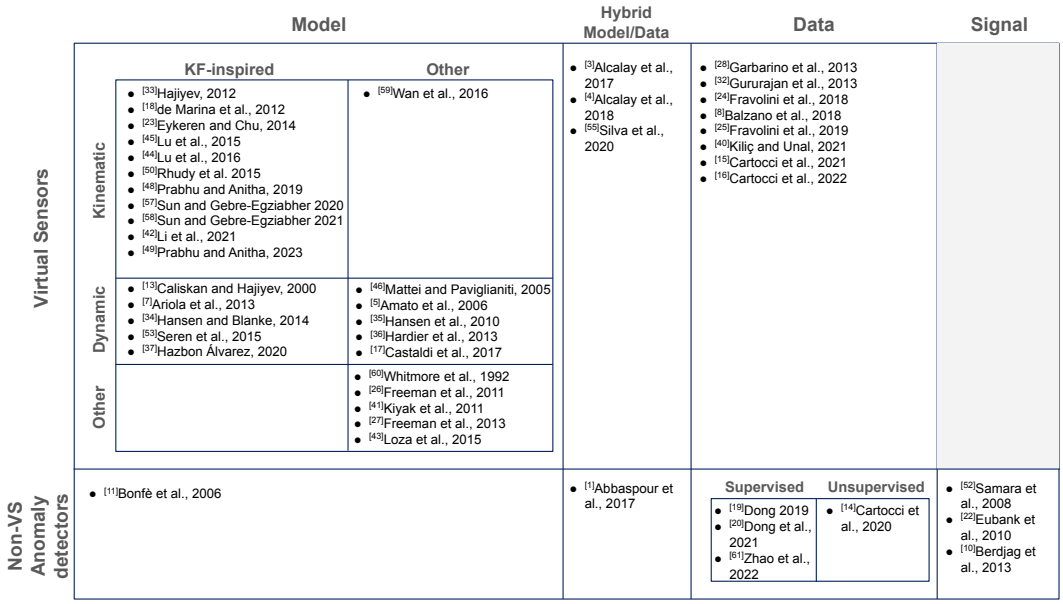


Figure 1 Clustering of different papers reviewed according to the defined classification criteria.

The temporal analysis of the reviewed articles reveal that model-driven approaches have received the highest attention throughout the period analyzed, while data-driven approaches have become popular in the last 6 years, likely motivated by the overall increase in popularity of machine learning and notably deep learning in the same period across domains.

3.2 Model-driven methods

Model-driven approaches rely on a model of the system to perform fault detection and isolation. Different authors impose various restrictions to delimit category. In this paper, this definition is relaxed to include not only parameter estimation and observer approaches, but also classical FDI techniques based on analytical redundancy relations. The idea is to highlight their common dependency on a set of equations that theoretically describe the system.

The second classification criteria presented previously was the separation between VS and non-VS anomaly detectors. The model based approaches are almost always VS approaches, performing detection by comparing the difference between observation and estimation, the so called residue.

The next classification criteria chosen was the physical model used by the proponents. Two main models were used, the aircraft flight kinetic model and the flight dynamics model.

- **Kinematic model**: a kinematic model is by definition a physical model of the system that concerns itself with movement with no regard to its cause, therefore not taking the forces involved into consideration. In the context of aircraft movement and air data sensors, this means a system of equations that associate airspeeds in the body-axis (u, v, w) , angle of attack (α), sideslip angle (β) and attitude angles (ϕ, θ, ψ) to ground speed (V_x, V_y, V_z), attitude angular rates around the body axis (p, q, r), accelerations (a_x, a_y, a_z), wind speeds (w_x, w_y, w_z) and true air speed (TAS). The exact equations used throughout the analysed papers vary according to objective and system specificity, but they derive from the model below with vectors on the body-frame system of coordinates.

$$\begin{cases} x = [u, v, w, \phi, \theta, \psi]^T , \\ z = [a_x, a_y, a_z, p, q, r]^T , \\ y = [TAS, \alpha, \beta, V_{xGPS}, V_{yGPS}, V_{zGPS}]^T \end{cases} \tag{1}$$

With state-transition equations given by:

$$\begin{cases} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} 0 & w & -v \\ -w & 0 & u \\ w & -u & 0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} - \begin{bmatrix} \dot{w_x} \\ \dot{w_y} \\ \dot{w_z} \end{bmatrix} \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & s_\phi \\ 0 & s_\phi c_\theta^{-1} & c_\phi c_\theta^{-1} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \end{cases} \tag{2}$$

And observation equations given by:

$$\begin{cases} TAS = \sqrt{u^2 + v^2 + w^2}, \\ \alpha = arcsin(w/u), \\ \beta = arcsin(v/u), \\ \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = DCM(\phi, \theta, \psi)^{-1} \left(\begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} \right) . \end{cases} \tag{3}$$

The kinematic model presents the advantage of simplicity. Accelerations and angular rates in the body frame of reference are given by the inertial reference system, while ground speeds are provided by GPS. No model of forces is required, therefore no aerodynamic equations are necessary. This simplicity is likely the reason why this is the most used

modeling in the papers reviewed. The ADS-derived observations monitored by this modeling are TAS, AoA and β. The immediate issue from using this approach is that a fault in the static pressure cannot be easily differentiated from a fault in the total pressure, as they are not individually monitored. Instead, a fault in any of these variables would affect the TAS observation. Most works that chose this approach had as main concern the diagnostic of fault in airspeed measurements as whole, instead of conceiving an individual monitoring of static and total pressure.

- **Dynamic model**: The flight dynamics model, on the other hand, concerns itself with the causes of the movement, therefore calculating the aerodynamic forces. As there are many different forms of adding force equations to the kinematic model in order to allow observability of P_s, P_t and TAT, a framework model will not be derived here as done with the kinematic model, but only descriptive explanation. Considering flight in subsonic compressible regime, the TAS can be associated to M and to T_s by $TAS = M \times c = \sqrt{\gamma R T_s}$ Where c is the speed of sound through air, R the atmospheric air individual gas constant and γ the heat capacity ratio. M on the other hand can be associated to P_s, P_t, T_s, T_t through the energy equation, one of its forms presented below:

$$\frac{P_t}{P_s} = \frac{T_t}{T_s}^{\gamma/(\gamma-1)} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\gamma/(\gamma-1)} \tag{4}$$

  Additional aerodynamic forces equations allow for the calculation of the dynamic pressure $P_d = P_t - P_s$, one these being the lift L equation $L = P_d S C_L$. Where S is the aerodynamic surface of the airplane and C_L the lift coefficient.
  With the kinematic equations providing acceleration, with the knowledge of the aircraft mass and using the aerodynamic forces equations, the dynamic pressure and consequently with the energy equations static pressure and total pressure become observable. A requirement for that is to have a state transition equation for the static temperature, or an equation that allows its observation from the internal states, this can be accomplished by the use of an stand atmospheric model or through the use of a propulsion model, which will not be discussed here.
  The dynamic model enables observability of total pressure, static pressure and total temperature sensors, but comes with added complexity. Surrogate models for aircraft aerodynamic coefficients are needed, as these vary with control surfaces configurations, incidence angles and flight regime. Modeling static temperature with a standard atmospheric model comes with significant imprecision, while using a propulsion model requires detailed engine performance knowledge. Reaching the expected accuracy may then require the estimation of modeling biases leading to a complex management of the model observability. This is likely the main reason why the use of dynamic equations is less popular in the reviewed papers.

The next classification criteria of the model-driven methods was with respect to the approach used for FDI.

The simplest form of model-driven FDI is through the use of analytical redundancy relations (ARR). An ARR can be obtained from a structurally overdetermined set of equations that include the system internal states and observations as variables. These can be obtained, for example, through the use of parity equations. This method is simple and reliable, but does not deal well with evolving systems, external disturbances and notably model uncertainty. [47]

A second form of model-based diagnosis is through the use of estimators. By estimating the model parameters, and thus the physical parameters, through the use of the observations, inputs and the model, one can observe the changes on these estimated parameters to declare

or not a fault. Notable techniques include recursive least squares estimation, the Hagglund method, and the Favier and Arruda method [47]. Estimators are another simple approach, but ignore any temporal evolution of the system states if they are available.

A most popular approach however is the construction of state observers. This approach exploits the knowledge of state-transition equations that provide predictions of the system next state, and of observation equations that associate these states at each instant to the available observations. [56]

In this review, we classify the model-driven approaches in two categories: Kalman filtering (KF) based approaches and non-Kalman filtering based (non-KF) approaches. This classification is motivated simply by the extensive representation of KF based approaches compared to others for ADS variables estimation and FDI.

3.2.1 KF-based approaches

The Kalman filter was initially proposed in [39]. Since then, it has been extensively used to filter noise of observations. The Kalman Filter is a statistical ponderator based in two steps:

- A **prediction step** where the internal states are propagated using a state-transition matrix and an observation matrix
- A **correction step** performed after the calculation of an optimal gain that will weight the contribution of prediction and observation in the final consolidated estimation.

The residues between predictions, observations and estimations can be used for FDI. The original KF formulation was only adapted to systems with linear models of state transition and observation equations and gaussian zero mean noises, which is not adapted for the models of ADS sensors observations described previously. Many derivative forms of the Kalman filter exist to overcome the mentioned limitations and relax the model hypothesis.

The Extended Kalman filter (EKF) is one of these derivative methods, and involves the linearization of the non-linear state-transition and observation equations through the calculation of the respective jacobians around the point of last estimation. In [23], authors apply the EKF on the kinematic model of the aircraft tested on a high fidelity simulator of a twin engine aircraft. The paper also proposes an exhaustive testing of faults with a Monte Carlo approach for fault inserting. The paper however assumes constant wind, a condition not usually seen in reality. [58] and [57] also propose the use of an EKF for the kinematic model, but have a weaker hypothesis on wind conditions, by assuming a first order Gauss-Markov model for wind speed, which supposes wind dynamics are slow, but not constant. More robust to slow wind gradients, but not to turbulence and strong gradients where the lack of a predictive wind model still makes the use of residues for fault diagnosis not robust enough to atmospheric conditions. The paper proposes several forms of FDI using this EKF, including a Receiver Autonomous Integrity Monitoring inspired statistical test for detection. [33] applies EKF to the kinematic model, but proposes an operative method of testing the innovation covariance of the KF, where the defined monitoring statistic is the maximal eigenvalue of the random Wishart matrix. The idea being that determining a confidence domain for testing in real time the KF innovation covariance is harder than determining it through the Tracy-Widom distribution.

[42] proposes a method for FDI on ADS based on EKF on the kinematic model, but considers the time offset between ADS and IRS sensor measurements as a source of possible errors that can trick the detection algorithm. The authors propose an additional observer, with a state-transition matrix defined as the identity matrix, that estimate the time-offset to be used on re-aligning measurements. Since the offset is only observable during maneuver,

their overall air data variables observers adapt during and out of maneuver time. [48] uses a variation of the EKF on the kinematic model of the aircraft, the Exponentially Weighted Adaptive Kalman Filter. This approach calculates the innovation covariance matrix as an exponentially decreasing weighted variance sum of past innovations using a fixed size time window to attenuate external disturbances and is supposed to provide more robust fault detection on turbulence situations. Wind however is modeled as a random walk process, not allowing for assessment of its robustness on strong wind gradients. A work from the same author takes a similar approach in [49], here however an Iterated Optimal Extended Kalman Filter (IOEFK) is used as a tentative to better accommodate the strong non-linearity of the physical model. The EKF works with local linearizations of the model equations, the IOEKF tries to improve the precision of these linearizations by iterating the point around which they are performed by performing the update step of the EKF multiple times.

Another form of dealing with the linear limitations of the KF is by adopting an ensemble strategy. By operating on ensembles of points following a distribution, one does not need to calculate expected values and variances directly. This is the case of the Unscented Kalman Filter (UKF), that uses a collection of sigma points, propagated using the direct non-linear equations of state-transition and observation. [18] uses two UKFs. The first estimates TAS and AoA using thekinematic equations under constant wind assumption and its residues are used for FD. A second UKF estimates the magnitude of the fault when it occurs by estimating the value of an added variable measuring the bias of AoA. The idea behind two filters is to use uncorrected estimations for FD and corrected estimations for fault characterization.

In [45] and [44], authors propose what is essentially an UKF applied to the kinematic model, which they name Robust Three Step Unscented Kalman Filter, for fault detection and isolation. Authors compare the performance of this filter against an Adaptive Fading Unscented Kalman Filter and apply both methods to real flight data. Although not mentioned, it seems wind was considered constant in this work, which is likely why the simple UKF did not perform well. Authors propose a modified version of it they name the Modified Robust Three-Step Unscented Kalman Filter, an UKF that performs a few measurement updates before operation to calibrate the initial states, which performs better. [51] also proposes the UKF applied to an UAV kinematic model, comparing its performance to that of the EKF, where only a slight difference in estimation performance was found in favor of the EKF. Author proposed cumulative sum control charts (CUSUM) and floating limiters for the FD step after obtaining innovations from the filters.

Many authors preferred the kinematic model approach presented above due to its simplicity and reduced model uncertainty. Those who used the dynamic model alternatively had to cope with its complexities in different forms. That is the case of the work of [34], where the aerodynamic lift force is calculated and a model of the engine is used for the calculation of thrust. To circumvent the lack of state-transition equations, the author simplifies the prediction step of the filter to a random walk process, sacrificing the information gained by the time propagation. Fault detection is achieved through a change detection framework through the use of Generalized Likelihood Ratio Tests (GLRT). This approach to dealing with the dynamic model complexity is somewhat similar to that undertaken by [3] and [4] where the state vector chosen is $X = (\overrightarrow{W}, b_{C_L}, \Delta_{ISA}, z_C)$, respectively wind speed vector, bias between lift coefficient from surrogate model and from lift equation, temperature deviation from standard atmosphere model and altitude deviation from standard atmosphere model. The estimation of this state vector allows for the estimation of modeling errors and the analysis of their magnitudes allows for the fault detection. The similarity with the previous work is that no state-transition equations are derived for the state prediction step, instead

a constant dynamic is assumed for all these variables, except for b_{C_L} for which a slow first order damping is assumed. [37] also proposes an EKF observer for the dynamic model of the aircraft, where residues are compared to a fixed threshold for fault detection. The interest in the author's approach is the definition of these thresholds as a gain multiplied by the mean residue, setting this gain as a compromise between false alarm rate and sensitivity to faults. [7] proposes the use of an EKF for the dynamic model but with the addition of a bank of integrators, one integrator per residue of each observation, used for fault detection and isolation. [53] proposes the use of an Adaptive Extended Kalman Filter for the observation of ADS variables, tuning the covariance error matrix of observations and state transitions in case of a detected fault. The downside of the proposed approach for larger airplanes is its reliance on a thrust model of the engines, which is not always reliable given the wide range of operation of commercial aircraft.

[13] takes a different perspective on the problem, and is instead concerned on defining the best metrics surrounding KF innovations and residues for optimal FDI on ADS. Four metric are compared, including the trace of the covariance matrix and the generalized variance of the covariance matrix of the innovation sequence, each of the 4 metric present a compromise between robustness to false alarms and sensitivity to faults.

3.2.2 Non KF-based approaches

Despite the majority of papers working with implementations of Extended Kalman Filters and Unscented Kalman Filters, a part of the reviewed works proposed non-KF based alternatives. In this section, these alternatives are presented, those comprise a range of methods from direct analytical redundancy relations to state observers not deriving from the KF.

[60] proposes an analytical redundancy approach to fault detection on a flush AoA probe of an F16 fighter jet. The non-linear model of the sensor is used to generate residues of the individual static pressure ports, and test these residues based on a Chi-square metric for fault detection. The non-linear model is linearized around the last non-failed state to allow the computation of its inverse functions. If fault is detected, a series of statistical tests are performed for isolation: including a reason check and a simple residue magnitude check. This is one example of model-driven approach applied neither to the kinematic nor dynamic models, but to the specific flush probe non-linear model. [17] also works on the direct analytic redundancy relations, by proposing the generation of residues that discriminate the occurrence of different faults so that isolation can be performed with the classical use of a fault sensitivity matrix, using non-linear geometric approach with singular perturbations (SP). SP theory allows for good residue generation on fast and slow dynamic points. Author considers altitude, speed and rate of climb to follow a slow dynamic while angle of attack and pitch rate follow a fast dynamic. [35] proposes another method of FDI on ADS not using observers for a small UAV. In it, residues are generated by using air data measured airspeed and airspeed obtained through the calculation of GPS ground speed (obtained through a KF). The wind speed is assumed to be constant and the airspeed is obtained through the use of an engine thrust model. Residues obtained are found to have colored noise, and filters tuned using flight data are used to bring their means to zero. A generalized likelihood ratio test is performed both on raw signal and whitened signal for FDI. [11] presents an elegant application of redundancy equations by generating residues using the aircraft dynamic model through the use of polynomial residue generators.

[36] proposes an estimator approach for the generation of redundant measurements and residues. FDI is based on the redundancy of aerodynamic coefficients provided by the use of the force and moment equations and the use of surrogate models for these coefficients.

The values of angle of attack and speed are calculated from a least squares optimization procedure. This optimization uses a second order local optimizer to minimize the error between surrogate coefficients and the coefficients coming from the aerodynamic equations. This is an interesting application of model inversion through optimization. However, it relies on the dynamic model, notably on a model for engine thrust, which as mentioned previously might limit its application.

Apart from the KF based observers, the most popular alternative filter used was the H_∞ or *Minmax* filter. In [26] and [27], the author proposes the use of the H_∞ filter for the problem of air data fault diagnosis and estimation applied to the NASA Generic Transport Model (GTM). The scope of the diagnosis is restricted to total pressure and static pressure faults of partial and full blockage. The choice of the H_∞ is interesting as it eliminates the need for strong hypothesis on the noises and disturbances. Another interesting contribution is the implementation of the approach on a closed loop context, where autopilot reference and control commands are used as observations, although authors claim no performance gains were found. [46] applies the H_∞ to the dynamic model of a small commercial aircraft. Here a bank of filters is used to isolate the fault, each bank estimating the observations of one of the two available ADRs.

Unknown Input Observers (UIO) were also present in the evaluated works. That is the case of [41], where an UIO is used to generate residues for threshold analysis for fault detection, with a generalized observer scheme (GOS), where each residue is sensitive to all faults except for one. In [5], authors propose a novel nonlinear observer exploiting UIO theory to guarantee disturbance decoupling with an H_∞ performance level.

Wind disturbances are a major concern for the design of a virtual sensor for ADS. Different approaches were presented previously, including simplifying hypothesis, constant or slow changing wind, or the inclusion of these as state variables. Other authors consider wind disturbances to be bound, using this property for designing observers. This is the case of [59], where the proposition is to perform a moving horizon state estimation using the kinematic model as an optimization problem using Karush-Kuhn-Tucker (KKT) conditions to deal with wind bound constraints, in order to increase the robustness to wind while maintaining sensitivity. [43] also considers the bounded wind hypothesis to estimate the faults based on a non-homogeneous high-order sliding mode observer, in finite time and in the presence of bounded disturbances. The sensor faults are estimated for the class of systems satisfying the structural property of strong observability. A key feature of the proposed solution is concerned by the effect that measurement noise could have on fault reconstruction. The physicial model used was obtained through linearization around a THS trimmed point, which limits however the solution proposed.

3.3 Data-driven methods

Data-driven methods have become increasingly popular for ADS FDI and for FDI globally as our review has revealed. The main challenge for the application of these techniques are the availability and quality of data. There are usually three sources of ADS data: simulation, flight test and operative flight data. Their use present specific challenges as described in table 1

The presented challenges related to the data use are maybe the main obstacles for the data-driven approaches on ADS FDI, but they are also a blocking point when evaluating and surveying papers, as the performance indicated by authors might often be distant to that achieved by a real application. Most of the data-driven papers reviewed have used simulation data, unless stated otherwise in the text.

Table 1 Description of the three sources of ADS data (simulation, operation and flight test) according to availability, quality, diversity and representativeness of data.

Source	Availability	Quality	Diversity	Representativeness
Simulation	Readily available and allowing the artificial creation of unlimited data	Sampling rates can be set at will, virtually any parameter of interest can be measured.	Depends on simulator complexity, overall many different weather conditions, operation points and maneuvers can be simulated.	Often important deviations from data expected in operation
Operation	Usually not publicly available, but available to airlines and often shared with manufacturers.	Sampling rates are often low, gaps not uncommon. Traceability of maneuvers and flight conditions often poor. Often not all features of interest are recorded.	Diversity of weather conditions, aircraft serial numbers and itineraries. Operation will not reach the extremes of aircraft capabilities and of flight envelope.	The highest possible, as it issues from the final application
Flight test	Industrial secret kept by aircraft manufacturers and not publicly available	High parameter coverage, high sampling rate, thorough annotation of the flight conditions and maneuvers.	Wide range of operation points tested. Aircraft evaluated to the limits of its flight envelope. Multiple allowed and forbidden maneuvers are tested.	Representative data. Limited to a few serial numbers, weather conditions and routes.

Following our defined classification, the existing data approaches are first divided into virtual sensors and non-VS Anomaly Detectors.

3.3.1 Data-driven virtual sensors

VS designs using data-driven methods consist of estimators trained usually on nominal non-faulty data. They are capable of estimating a variable of interest at an instant of time using the other available observations. The observations used might correspond to the current instant only, or can additionally be composed of previous observations along a fixed or variable time window. The previous observations of the variable of interest itself can be used for the estimation. Multiple detection and isolation techniques can be used once the estimations are available, and are usually independent of the estimator.

The simplest form of data-driven VS uses only current observations for estimating the evaluated variable, exploiting implicit relationships between different observations. This is analogous to identifying analytic redundancy relationships using a system model, the difference here being that the system equations are not known and we would like to extract them from system operation data. The advantage of this approach is that many simple machine learning and regression models can be used, feed-forward neural networks (FFNN) of varying architectures, linear and non-linear regressors, regression splines, regression trees, bayesian regression are some examples. The weakness of these methods is that the equation parameters that model the aircraft dynamics are not steady, but instead evolve with fuel consumption and with the changes of aircraft configurations, e.g. high and low lift configurations. If enough diversity of data is available, the simple model should be able to take these as features as well, however the atmospheric disturbance that effect the equations cannot generally be measured, and these simple approaches will suffer from this imprecision.

[28] is one of the earliest works on the use of data-driven methods for ADS FDI. The author uses a simple feed-forward neural network to produce redundant estimations of indicated airspeed using other ADS measurements along with GPS data. The downside is that its real

applicability is very limited, since no consideration was given to wind disturbances. [32] takes a similar approach, estimating airspeed through the use of IRS data, in this case testing two FFNN architectures, a first basic Multi-Layer Perceptron (MLP) and an Extended Minimal Resource Allocating Neural Network. Again no wind corrections were considered. Furthermore as train and validation sets were extracted from data points of the same flights, doubts can arise on its claimed performance.

The work of [8] is also included in this time-less regression approach. Here, a series of neural networks are used to estimate TAS, AoA and β. The main innovation comes on the residue-threshold analysis, where the thresholds are not fixed but are trained using real flight data. The threshold that limits normal operation is therefore dependent on the operating point of the aircraft. [40] proposes another regressor without past observations, but based on Artificial Neural Fuzzification Inference (ANFI) and MLP. The interesting point in the author's approach is that the thresholds for detection were not based on the regressor performance itself, but on the Federal Aviation Agency (FAA) acceptable deltas between captain and first officer displays for the different variables. Features were selected for the regression based on simple Pearson correlation.

Authors also have experimented with simpler regressors, as shown by [15], where linear regression is used to estimate the interest variables the observations of the other sensors and the control inputs. Isolation using residues is done in different forms: the first is by using Mahalanobis distance to compare the residue vector to the fault directions. Another form was by reconstructing the fault vector by finding the minimal fault that best approximates the corresponding residue to the real one, based on the hypothesis that only one fault occurs at a time. Using linear regressions to capture the implicit relations in a non-linear system presents obvious limitations, and the same author proposes an expansion to this work in [16], where non-linear additive regression is used. The author takes advantage of the additive nature of this approach to derive the fault-sensitivity matrix of the observations by local linearization using the first order term of the Taylor expansion of the non-linear terms in function of the states. The author also proposes to take advantage of analytical redundancy through assumed non-faulty variables to produce estimates of the observations through mutivariate adaptative regression splines. The residue is then used to detect a fault.

[24] takes a different and interesting approach, by using interval models to deal with uncertainty on estimation. measurements that fall outside of the interval bounds are considered faults. The non-linear dynamics are estimated by a neural network, while the interval model, linear in nature, explains the dynamic noise and variation.

The next form of data-driven VS can be achieved using the time information provided by the previous observations, this is the data-driven approach analogous to the construction of an observer in a model-driven approach. This approach requires the use of more complex models. The classes of neural networks with memory capabilities, such as Recursive Neural Networks (RNNs), Long-Short Term Memory Units (LSTMs) and Transformers are a few examples. Autoregressive models, e.g. Non Linear Autoregression with Exogeneous Variables (NARX), and time series models that consider exogenous inputs can also be used, such as Autoregressive Integrated Moving Average with Exogenous Inputs (ARIMAX).

This has been the least used approach for data-driven VS design in ADS as observed in this review. [25] tests a series of adaptative regression algorithms for the estimation of TAS and subsequent FDI. The non-adaptative models are the classic implementations of NARX and of the Linear Finite-Impulse Response Model, while their adaptative versions add a term that tends to converge the estimated value to the measured value, analogous to an update step for a KF. The author finds that the adaptative versions of these algorithms performed better and claims that method can detect faults on TAS of as little as 0.7 m/s while being robust to false alarms.

3.3.2 Data-driven non-VS anomaly detectors

Anomaly detection in the context of data analysis is the identification and isolation of items, events or data points that are rare, or that deviate from the trends observed for the majority of the data or, the definition that is more important in our context, that do not follow what is considered as the normal behavior. In the context of ADS FDI, nominal behavior can be seen as sensors measuring their respective values within expected levels of precision, defined often by the sensor uncertainty informed by the manufacturer or by the needs of autoflight, flight control, navigation and guidance. Three types of training for non-VS anomaly detectors are differentiated: supervised, semi-supervised and unsupervised.

Supervised trained non-VS anomaly detectors use training data that is labeled with fault or no-fault. The fault label can be further characterized by the type or class of fault, in which case a classification is performed for a multiclass anomaly detection. This will often be the case with simulated data, where faults are ingested artificially and precise labeling is possible.

[20] and [61] frame the FDI problem similarly as a supervised classification, performed by a Deep Neural Network (DNN) with convolutional layers and LSTM units. 2 dimensional inputs are used, a dimension for the different features and one for the different flight data points in a fixed size time window. The convolutional layers and LSTMs units are used to capture the patterns, such as drift, present in the data that can reveal a fault. A dictionary of 5 types of fault is created labeled from 1-5, while the label 0 is given to nominal flight points. An interesting approach, but limited by the need of defining a dictionary of faults in advance. [19] also chooses to use a supervised trained DNN approach, but focused on the cases of icing detection, an interesting advance is the use of transfer learning techniques to adapt the trained model to the similar task of icing detection on actuators.

Semi-supervised trained non-VS anomaly detectors have a partially labeled data. In the case of ADS FDI, this means having a collection of data points known to be nominal, but no faulty data points. This type of models is often suitable for training using real flight test data. The quality control of flight tests ensure the data is nominal, but no faulty data is usually available for training. Notable exceptions exist, namely when a flight test is performed to test aircraft robustness to a specific type of fault or when faults are artificially inserted into the nominal flight test data after its execution. The former produces only limited cases of fault data while the latter cannot guarantee that the inserted faults represent well the true faulty behavior mid-flight, hindering the use of supervised approaches efficiently. This approach has been largely ignored for ADS FDI to this moment, and is a probable pathway for future works. One reason for this being the use of simulated data by most of the papers reviewed, which is more prone to the training of supervised methods. Readers can refer to [50] for example, where autoencoders are used for FDI of electric motors. Autoencoders are a class of semi-supervised anomaly detectors. They are trained to reduce the number of dimensions of nominal data through encoding, bringing data to a latent representation but retaining a maximum of information. This retaining allows a subsequent decoding of this latent representation back to the normal representation with minimal errors. Ideally, an autoencoder trained in this manner would have low performance when encoding and decoding anomalous or faulty data, as the implicit relations between the dimensions that allow for the retaining of information when bringing data to a latent representation would not be respected. This difference of performance allows for fault detection and isolation.

Unsupervised training is the most challenging form of producing an anomaly detector, and also the most classic. Data is not labeled, and algorithms will try to separate the observations into groups, often referred as clusters, following underlying data patterns. Some

interpretation is usually needed for understanding which cluster correspond to normal data and which to do not. This is a less seen paradigma for ADS FDI, but can be the case when training is performed with aircraft operation flight data, where no nominal behavior is assumed.

Principal Components Analysis can be used for unsupervised anomaly detection, and is proposed by [14] for dimensionality-reduction of flight data. Metrics of distance are used inside the principal components space to define if a fault exists or not with a drift diffusion model to avoid fluctuations.

3.4 Signal-driven methods

Mainly signal-driven approaches for fault detection were not featured extensively in recent research ADS FDI, which is partly due to how consolidated these methods are, and the fact that they are already extensively used in aircraft as health checks.

[52] proposes a statistical method capable of independently monitoring a single sensor, without relying on additional redundancy or other variables. It involves cleaning the signal of nonstationary components, whitening via a proposed pooled autoregressive modeling, statistical decision making, and an electronic spike/glitch removal logic.

In [22], authors propose a diagnostic for an UAV seaplane with history of Pitot faults due to blockage with water. They privilege time-domain analysis to frequency-domain as they claim that the common classes of air-data sensors do not generally have a strong frequency component. The detection strategy is based on the analysis of multiple features extracted from sensor signal. The features verified are signal saturation, signal variance peak change rate, signal variance deviation threshold. The thresholds used to characterize a fault through the calcualted features are extracted from data. Fault is declared by a weighted average of the probabilities of fault given by each of the three signal tests, with higher weight for saturation.

[10] proposes a physical redundancy based FDI approach that borrows from data-driven and signal-driven methods. The work is focused on improving diagnosis when three redundant sources are available and allow for oscillatory fault detection and isolation when only two sources are available. The author proposes a fuzzy consolidation of the sources measurements when the three are available, allowing for a continuous transition for a valid consolidated measurement when one of them is to be rejected. Oscillatory fault detection is performed through the use of a Kalman filter to isolate the oscillatory component of the signal, transforming it into a step-like signal for residue-threshold comparison.

3.5 Hybrid methods

Hybrid methods are those that rely on a combination of partial/complete modeling knowledge of the system and data regarding its operation. Four common applications of hybridization are highlighted:

- **Data for compensation of non-modeled effects**: data-driven approaches used in series with model-driven approaches to capture effects not covered by the available model.
- **Model derivation**: when a physical model of the system is unknown, data-driven methods can be used for their derivation. One example is the use of symbolic regression to extract equations linking system variables.
- **Noise handling**: when the model uncertainty or the observation noises covariance matrix are unknown, or change with time/external variables, data-driven methods can be used to provide a noise estimate for the model-driven approach.

- **Adaptive threshold calculation** data-driven methods are trained to adapt the threshold for fault detection over the residues dynamically, when these change over time or by external action. This is done to improve sensitivity and to increase robustness to false alarms.

[55] adopts both the model derivation approach for the data/model hybridization and the adaptive threshold calculation. In the work, a Kalman Filter is used as observer, the model however is not explicitly provided but rather derived from flight test time-series data by means of dynamic mode decomposition (DMD). The threshold for fault detection on residue is decided through the use of a decision tree, trained on labeled faulty/non-faulty data. [4] uses a surrogate model for the lift coefficient trained as basic multi-layer perceptron to account for the unknown modeling, necessary for the implementation of the EKF. [1] proposes on the other hand a combination of EKF with a neural network to perform fault estimation, having its weights adapted online during aircraft operation.

4 Limitations of SoA and ways forward

Excellent works have been produced on the topic of ADS fault detection and isolation. The main problems tackled when applying FDI methods to ADS are the following:

- The underlying models are non-linear by nature,
- Sensors are exposed to external disturbances due to wind gradients and other phenomena, system and environment changing conditions, causing modeling uncertainties, especially when working with aerodynamic forces,
- A variety of types and modes of faults exist, some modes being discovered only after being observed in operation,
- FDI must be performed online on aeronautical-graded computers, with limited computational and parallelization power,
- FDI methods are required to have a high robustness to false alarms as aviation is a safety critical domain.

Signal approaches are useful on their simplicity and reliability to detect deviations from normal behaviour, and are currently used in airplanes not only for ADS FDI but for fault detection in actuators and other systems. Despite their usefulness, they often do not provide enough sensitivity to faults and they should not be used as the sole means for analytical redundancy.

Model-driven approaches can provide the required additional sensitivity and are a way to obtain virtual measurements of faulty variables allowing continued operation. Drawbacks exist however: the non-linear models demand adaptations of the observers used, which increases imprecision, notably seen in case of local linearization, such as with the Extended Kalman Filter and mitigated to some extent with ensemble strategies, such as used by the Unscented Kalman Filter and the Ensemble Kalman Filter. Another drawback is the lack of reliable modeling of the systems. This is most noticeable when methods are transported from the realm of simulation to the realm of real flight: dynamic models suffer from important deviations on the aerodynamic coefficients and on the engine behavior. Models are required to be adaptive or often simplified and some equations discarded. Kinematic and dynamic models suffer with the unknown wind variations, which was dealt with by authors in multiple ways, from bounded optimizations to simplification hypothesis, with varying degrees of robustness to real operation.

We believe data-driven approaches and hybrid approaches have the power to overcome some of these difficulties, exhibiting enough complexity so that they can be trained to detect small deviations but at the same time enough flexibility as to adapt to different flight

operation points with the necessary robustness, provided that the data presented for training has enough quality and diversity. Most notably, we believe semi-supervised approaches should be explored. As air data sensors are exposed, faults can have many different causes and present themselves in non predicted forms. This is why we believe practical implementations of supervised methods might suffer from the lack of generality, as labeled faulty data will hardly be representative enough in simulated or real data.

Both data-driven virtual sensors and semi-supervised anomaly detectors exploit nominal data, available with great diversity nowadays from simulated and real flight test campaigns. The former can work as a self-contained system, providing estimations and FDI. The latter, can be combined with model based observers or estimators to provide the estimations in case no trusted measurements are available. Possibilities for implementing data-driven VS are the use of NN ,RNNs, LSTMs and CNNs trained on nominal data to provide estimations. Training can be further improved through the use of data assimilation frameworks [12]. As for semi-supervised anomaly detectors: PCA, correlation analysis and autoencoders are simple but effective choices. These consolidated technologies are the starting points for the research of the authors of this paper.

One of the main historical challenges concerning the use of data-driven methods in aeronautical applications has been the certification. The current trend is for this issue to be mitigated due to the evolution of regulations around the world. The European Union Aviation Safety Agency (EASA) has recently issued a concept paper with guidance for future aeronautical machine learning applications indicating four building blocks for the framework of a *trustworthy AI*: AI trustworthiness analysis, AI assurance, human factors for AI and AI safety risk mitigation [21]. This hints to the future progress and wide utilization of more AI based solutions, which further corroborates our recommendation.

Another important challenge to data-driven methods is explainability and interpretability. An explainable AI model is one whose inner workings can be explained in a human comprehensible manner. An interpretable model is one for which the effects on the outputs through the variations of its inputs are known [30]. Interpretability can be attained more often than explainability. Model-driven and signal-driven approaches are explainable and interpretable, which is often not the case for data-driven approaches. Trustworthy AI will require a degree of interpretability and explainability that is yet to be addressed, as virtually none of the papers reviewed tackled the topic.

Another point of improvement for future works concerns the analysis of robustness. As previously mentioned, false alarms are not tolerated on aeronautical applications. This is especially true at certain times, such as take-off and landing, during turbulence or strong gusts of wind. Therefore, an assessment of robustness should include not only the rates of false alarm, but also an analysis of when they are most prevalent.

5 Conclusions

This review paper intends to perform an exhaustive review of publicly available papers on embedded FDI methods of air data sensors. The papers according to different criteria have been grouped, finding that most proposals were model-driven virtual sensors based on KF variations, most notably the Extended Kalman Filter. Furthermore, it was found that data-driven methods have gained popularity in recent years, with a majority of those works focusing on virtual sensors based on non-linear regressions and different types of neural networks.

We present the challenges for ADS FDI, and highlight the trend towards data-driven methods and the availability of data due to extensive flight test campaigns performed throughout the decades and due to more accurate simulations. Due to those, we believe an important way forward to ADS FDI research lies on data-driven methods, in special on non-VS anomaly detector architectures. These have not received enough attention, as observed in the review, but are especially interesting in ADS systems where often double redundancy is available and the bottleneck for the system lies on the selection of one working sensor alongside two faulty sensors. Certification for these types of systems is still a major challenge as they often have low degree of explainability and interpretability.

References

1 Alireza Abbaspour, Payam Aboutalebi, K. Yen, and A. Sargolzaei. Neural adaptive observer-based sensor and actuator fault detection in nonlinear systems: Application in UAV. *ISA transactions*, 67:317–329, 2017. doi:10.1016/j.isatra.2016.11.005.

2 'AIRBUS'. Lightning strikes. https://safetyfirst.airbus.com/lightning-strikes/. Accessed: 2024-09-27.

3 Guillaume Alcalay, Cédric Seren, Georges Hardier, Martin Delporte, and Philippe Goupil. Development of virtual sensors to estimate critical aircraft flight parameters. *IFAC-PapersOnLine*, 50(1):14174–14179, 2017. Number: 1 Publisher: Elsevier.

4 Guillaume Alcalay, Cédric Seren, Georges Hardier, Martin Delporte, and Philippe Goupil. An adaptive extended kalman filter for monitoring and estimating key aircraft flight parameters. *IFAC-PapersOnLine*, 51(24):620–627, 2018. Number: 24 Publisher: Elsevier.

5 Francesco Amato, Carlo Cosentino, Massimiliano Mattei, and Gaetano Paviglianiti. A direct/functional redundancy scheme for fault detection and isolation on an aircraft. *Aerospace Science and Technology*, 10(4):338–345, 2006. Publisher: Elsevier.

6 John Anderson. *EBOOK: Fundamentals of Aerodynamics (SI units)*. McGraw hill, 2011.

7 Marco Ariola, F Corraro, Massimiliano Mattei, I Notaro, and Adolfo Sollazzo. An SFDI observer-based scheme for a general aviation aircraft. In *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 152–157. IEEE, 2013.

8 Fabio Balzano, Mario L Fravolini, Marcello R Napolitano, Stéphane d'Urso, Michele Crispoltoni, and Giuseppe del Core. Air data sensor fault detection with an augmented floating limiter. *International Journal of Aerospace Engineering*, 2018(1):1072056, 2018. Publisher: Wiley Online Library.

9 Joelle Barthe. Unreliable speed. https://safetyfirst.airbus.com/app/themes/mh_newsdesk/documents/archives/unreliable-speed.pdf. Accessed: 2024-09-27.

10 Denis Berdjag, Jerome Cieslak, and Ali Zolghadri. Fault detection and isolation of aircraft air data/inertial system. *EUCASS Proceedings Series-Advances in AeroSpace Sciences*, 6:317–332, 2013.

11 Marcello Bonfè, P. Castaldi, W. Geri, and S. Simani. Fault detection and isolation for on-board sensors of a general aviation aircraft. *International Journal of Adaptive Control and Signal Processing*, 20, 2006. doi:10.1002/ACS.906.

12 Pierre Boudier, Anthony Fillion, Serge Gratton, Selime Gürol, and Sixin Zhang. Data Assimilation Networks, May 2023. arXiv:2010.09694 [cs, eess]. URL: http://arxiv.org/abs/2010.09694.

13 Fikret Caliskan and Chingiz M Hajiyev. Innovation sequence application to aircraft sensor fault detection: comparison of checking covariance matrix algorithms. *ISA transactions*, 39(1):47–56, 2000. Publisher: Elsevier.

14 N. Cartocci, G. Costante, M. Napolitano, P. Valigi, F. Crocetti, and M. L. Fravolini. PCA Methods and Evidence Based Filtering for Robust Aircraft Sensor Fault Diagnosis. *2020 28th Mediterranean Conference on Control and Automation (MED)*, pages 550–555, 2020. doi:10.1109/MED48518.2020.9182973.

15 N. Cartocci, M. Napolitano, G. Costante, and M. L. Fravolini. A Comprehensive Case Study of Data-Driven Methods for Robust Aircraft Sensor Fault Isolation. *Sensors (Basel, Switzerland)*, 21, 2021. doi:10.3390/s21051645.

16 Nicholas Cartocci, Marcello R. Napolitano, Francesco Crocetti, Gabriele Costante, Paolo Valigi, and Mario L. Fravolini. Data-Driven Fault Diagnosis Techniques: Non-Linear Directional Residual vs. Machine-Learning-Based Methods. *Sensors*, 22(7), 2022. doi:10.3390/s22072635.

17 Paolo Castaldi, Nicola Mimmo, and Silvio Simani. Avionic air data sensors fault detection and isolation by means of singular perturbation and geometric approach. *Sensors*, 17(10):2202, 2017. Publisher: MDPI. doi:10.3390/S17102202.

18 García H de Marina, Andres Marcos, and Rodrigo Haya. Angle of Attack and True Airspeed failure sensor detection and recovery based on Unscented Kalman Filters for the ALPHA vehicle. *Ifac Proceedings Volumes*, 45(20):1197–1202, 2012. Publisher: Elsevier.

19 Yiqun Dong. Implementing Deep Learning for comprehensive aircraft icing and actuator/sensor fault detection/identification. *Eng. Appl. Artif. Intell.*, 83:28–44, 2019. doi:10.1016/J.ENGAPPAI.2019.04.010.

20 Yiqun Dong, Jiongran Wen, Youmin Zhang, and Jianliang Ai. Deep neural networks-based air data sensors fault detection for aircraft. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 442–447. IEEE, 2021.

21 European Union Aviation Safety Agency EASA. Easa concept paper: First usable guidance for level 1&2 machine learning applications, 2023.

22 Ryan Eubank, Ella Atkins, and Stephanie Ogura. Fault detection and fail-safe operation with a multiple-redundancy air-data system. In *AIAA Guidance, Navigation, and Control Conference*, page 7855, 2010.

23 L. V. Eykeren and Q. Chu. Sensor fault detection and isolation for aircraft control systems by kinematic relations. *Control Engineering Practice*, 31:200–210, 2014. doi:10.1016/J.CONENGPRAC.2014.02.017.

24 M. L. Fravolini, M. Napolitano, G. D. Core, and Umberto Papa. Experimental interval models for the robust Fault Detection of Aircraft Air Data Sensors. *Control Engineering Practice*, 2018. doi:10.1016/J.CONENGPRAC.2018.07.002.

25 Mario L. Fravolini, Giuseppe Del Core, Umberto Papa, Paolo Valigi, and Marcello R Napolitano. Data-Driven Schemes for Robust Fault Detection of Air Data System Sensors. *IEEE Transactions on Control Systems Technology*, 27:234–248, 2019. doi:10.1109/TCST.2017.2758345.

26 Paul Freeman, Peter Seiler, and Gary J Balas. Robust fault detection for commercial transport air data probes. *IFAC Proceedings Volumes*, 44(1):13723–13728, 2011. Publisher: Elsevier.

27 Paul Freeman, Peter Seiler, and Gary J Balas. Air data system fault modeling and detection. *Control Engineering Practice*, 21(10):1290–1301, 2013. Publisher: Elsevier.

28 Luca Garbarino, Gaetano Zazzaro, Nicola Genito, Giancarmine Fasano, and Domenico Accardo. Neural network based architecture for fault detection and isolation in air data systems. In *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, pages 2D4–1. IEEE, 2013.

29 Roger W Gent, Nicholas P Dart, and James T Cansdale. Aircraft icing. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 358(1776):2873–2911, 2000.

30 Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

31 Philippe Goupil. AIRBUS state of the art and practices on FDI and FTC in flight control system. *Control Engineering Practice*, 19(6):524–539, 2011. doi:10.1016/j.conengprac.2010.12.009.

32 Srikanth Gururajan, Mario L. Fravolini, Haiyang Chao, Matthew Rhudy, and Marcello R. Napolitano. Performance evaluation of neural network based approaches for airspeed Sensor Failure Accommodation on a small UAV. In *21st Mediterranean Conference on Control and Automation*, pages 603–608, 2013. doi:10.1109/MED.2013.6608784.

33 Ch Hajiyev. Tracy–Widom distribution based fault detection approach: Application to aircraft sensor/actuator fault detection. *Isa Transactions*, 51(1):189–197, 2012. Publisher: Elsevier.

34 Soren Hansen and Mogens Blanke. Diagnosis of Airspeed Measurement Faults for Unmanned Aerial Vehicles. *IEEE Transactions on Aerospace and Electronic Systems*, 50(1):224–239, 2014. doi:10.1109/TAES.2013.120420.

35 Søren Hansen, Mogens Blanke, and Jens Adrian. Diagnosis of uav pitot tube defects using statistical change detection. *IFAC Proceedings Volumes*, 43(16):485–490, 2010. Publisher: Elsevier.

36 Georges Hardier, Cédric Seren, Pierre Ezerzere, and G Puyou. Aerodynamic model inversion for virtual sensing of longitudinal flight parameters. In *2013 conference on control and fault-tolerant systems (SysTol)*, pages 140–145. IEEE, 2013.

37 Omar Hazbon Álvarez. *Fault tolerant air data system for pitot tube failure*. PhD thesis, Publisher: Escuela de Ingenierías, 2020.

38 Maria JESUS DE LA FUENTE. Foundations for fault detection and diagnosis in fdi and dx, June 2024.

39 R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960. doi:10.1115/1.3662552.

40 Ugur Kilic and Gulay Unal. Aircraft air data system fault detection and reconstruction scheme design. *Aircraft Engineering and Aerospace Technology*, 93(6):1104–1114, 2021. Publisher: Emerald Publishing Limited.

41 E. Kiyak, A. Kahvecioğlu, and F. Caliskan. Aircraft Sensor and Actuator Fault Detection, Isolation, and Accommodation. *Journal of Aerospace Engineering*, 24:46–58, 2011. doi:10.1061/(ASCE)AS.1943-5525.0000052.

42 Zhenwei Li, Yongmei Cheng, Huibin Wang, and Huaxia Wang. Fault detection approach applied to inertial navigation system/air data system integrated navigation system with time-offset. *IET Radar, Sonar & Navigation*, 15(9):945–956, 2021. Publisher: Wiley Online Library.

43 A. F. Loza, J. Cieslak, D. Henry, J. Dávila, and A. Zolghadri. Sensor fault diagnosis using a non-homogeneous high-order sliding mode observer with application to a transport aircraft. *Iet Control Theory and Applications*, 9:598–607, 2015. doi:10.1049/IET-CTA.2014.0226.

44 Peng Lu, Laurens Van Eykeren, E Van Kampen, CC De Visser, and QP Chu. Adaptive three-step Kalman filter for air data sensor fault detection and diagnosis. *Journal of Guidance, Control, and Dynamics*, 39(3):590–604, 2016. Publisher: American Institute of Aeronautics and Astronautics.

45 Peng Lu, Laurens Van Eykeren, Erik-Jan Van Kampen, and Q Ping Chu. Air data sensor fault detection and diagnosis with application to real flight data. In *AIAA Guidance, Navigation, and Control Conference*, page 1311, 2015.

46 Massimiliano Mattei and Gaetano Paviglianiti. Managing sensor hardware redundancy on a small commercial aircraft with Hinf FDI observers. *IFAC Proceedings Volumes*, 38(1):347–352, 2005. Publisher: Elsevier.

47 Ron J Patton and Carlos J Lopez-Toribio. Artificial intelligence approaches to fault diagnosis. In *IEE Colloquium on Update on Developments in Intelligent Control (Ref. No. 1998/513)*, pages 3–1. IET, 1998.

48 S. Prabhu and G. Anitha. An innovative analytic redundancy approach to air data sensor fault detection. *The Aeronautical Journal*, 124:346–367, 2019. doi:10.1017/aer.2019.143.

49 S. Prabhu and G. Anitha. Robust fault detection and diagnosis of primary air data sensors in the presence of atmospheric turbulence. *The Aeronautical Journal*, 2023. doi:10.1017/aer.2023.32.

50 E. Principi, D. Rossetti, S. Squartini, and F. Piazza. Unsupervised electric motor fault detection by using deep autoencoders. *IEEE/CAA Journal of Automatica Sinica*, 6:441–451, 2019. doi:10.1109/JAS.2019.1911393.

51 Matthew B. Rhudy, Mario L. Fravolini, Yu Gu, Marcello R. Napolitano, Srikanth Gururajan, and Haiyang Chao. Aircraft model-independent airspeed estimation without pitot tube measurements. *IEEE Transactions on Aerospace and Electronic Systems*, 51(3):1980–1995, 2015. doi:10.1109/TAES.2015.130631.

52 Paraskevi A Samara, George N Fouskitakis, John S Sakellariou, and Spilios D Fassois. A statistical method for the detection of sensor abrupt faults in aircraft control systems. *IEEE Transactions on Control Systems Technology*, 16(4):789–798, 2008. Publisher: IEEE. doi:10.1109/TCST.2007.903109.

53 Cédric Seren, Pierre Ezerzere, and Georges Hardier. Model–based techniques for virtual sensing of longitudinal flight parameters. *International Journal of Applied Mathematics and Computer Science*, 25(1):23–38, 2015. doi:10.1515/AMCS-2015-0002.

54 Sandeep Kumar Shukla and Jean-Pierre Talpin. *Synthesis of embedded software: frameworks and methodologies for correctness by construction*. Springer Science & Business Media, 2010.

55 Brian M. de Silva, Jared L. Callaham, Jonathan Jonker, Nicholas Goebel, Jennifer Klemisch, Darren McDonald, Nathan S. Hicks, J. Kutz, S. Brunton, and A. Aravkin. Physics-informed machine learning for sensor fault detection with flight test data. *ArXiv*, abs/2006.13380, 2020. arXiv:2006.13380.

56 Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

57 Kerry Sun and Demoz Gebre-Egziabher. A fault detection and isolation design for a dual pitot tube air data system. In *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 62–72. IEEE, 2020. doi:10.1109/PLANS46316.2020.9110179.

58 Kerry Sun and Demoz Gebre-Egziabher. Air data fault detection and isolation for small UAS using integrity monitoring framework. *Navigation*, 68(3):577–600, 2021. Publisher: Wiley Online Library.

59 Yiming Wan, Tamas Keviczky, and Michel Verhaegen. Robust air data sensor fault diagnosis with enhanced fault sensitivity using moving horizon estimation. In *2016 American control conference (ACC)*, pages 5969–5975. IEEE, 2016. doi:10.1109/ACC.2016.7526606.

60 Stephen Whitmore, Timothy Moes, and Cornelius Leondes. Failure detection and fault management techniques for flush airdata sensing systems. In *30th Aerospace Sciences Meeting and Exhibit*, page 263, 1992.

61 Yunmei Zhao, Hang Zhao, Jianliang Ai, Yiqun Dong, and others. Robust data-driven fault detection: An application to aircraft air data sensors. *International Journal of Aerospace Engineering*, 2022, 2022. Publisher: Hindawi.

A Study on Redundancy and Intrinsic Dimension for Data-Driven Fault Diagnosis

Daniel Jung
Linköping University, Sweden

David Axelsson
Linköping University, Sweden

Abstract

Data-driven fault diagnosis of technical systems use training data from nominal and faulty operation to train machine learning models to detect and classify faults. However, data-driven fault diagnosis is complicated by the fact that training data from faults is scarce. The fault diagnosis task is often treated as a standard classification problem. There is a need for methods to design fault detectors using only nominal data. In model-based diagnosis, the ability construct fault detectors depends on analytical redundancy properties. While analytical redundancy is a model property, it describes the diagnosability properties of the system. In this work, the connection between analytical redundancy and the distribution of observations from the system on low-dimensional manifolds in the observation space is studied. It is shown that the intrinsic dimension can be used to identify signal combinations that can be used for constructing residual generators. A data-driven design methodology is proposed where data-driven residual generators candidates are identified using the intrinsic dimension. The method is evaluated using two case studies: a simulated model of a two-tank system and data collected from a fuel injection system. The results demonstrate the ability to diagnose abnormal system behavior and reason about its cause based on selected signal combinations.

2012 ACM Subject Classification Computing methodologies

Keywords and phrases Data-driven diagnosis, intrinsic dimension, model-based diagnosis, structural methods

Digital Object Identifier 10.4230/OASIcs.DX.2024.4

Funding *Daniel Jung*: D. Jung is partially funded by the Swedish excellence center ELLIIT.

1 Introduction

The main objective of fault diagnosis of technical systems is detecting abnormal behavior and identifying its cause. Different scientific communities have approached this problem, e.g. model-based diagnosis and data-driven diagnosis. The general principle is to compare observations with model predictions to detect inconsistencies due to faults. When the abnormal behavior is significant compared to prediction inaccuracies, a fault is detected. After a fault has been detected, fault isolation is performed to compute diagnoses that are consistent with the observations. Model-based diagnosis relies on mathematical models derived from physical insights about the system to compute residuals. Data-driven diagnosis uses training data from relevant fault scenarios to learn the relationship between observations and diagnoses, e.g. using a classifier.

Having access to informative measurements from the system, e.g. sensor and actuator signals, is necessary to draw conclusions about the system's health. Model inaccuracies result in prediction errors or misclassifications which complicate fault detection and isolation. Methods are needed to systematically construct fault detectors based on the measurements. In model-based diagnosis, the concept of redundancy is central for design and analysis of diagnosis systems, see e.g. [17]. Analytical redundancy is necessary to construct residual generators, i.e., to formulate a mathematical model describing the relationship between a set


© Daniel Jung and David Axelsson;
licensed under Creative Commons License CC-BY 4.0



of known signals. Fault detectability and isolability properties of the model depends on what type of residual generators can be constructed [27]. If faults are not detectable or isolable, more sensors can be added to the system to improve the redundancy properties [18].

In data-driven diagnosis, the fault diagnosis problem is often treated as a general classification problem and the objective is to extract a set of features from the observations and train a model that can distinguish between different fault classes [24]. However, redundancy properties are, in general, not considered in the data-driven diagnosis system design process. If training data is representative of all fault classes, and the set of features is sufficiently rich, a data-driven classifier is expected to find a good separation between the different classes. However, if training data is not representative of the relevant fault scenarios, there is a significant risk that out-of-distribution samples, i.e. samples that deviate from operating conditions covered in training data, will result in false alarms and misclassifications [31].

Data-driven diagnosis have been used in various applications and different survey papers try to summarize the current state-of-the-art, see e.g. [1, 30, 29]. The survey paper [26] focused on machine learning for predictive maintenance. The importance of access to training data for data-driven fault diagnosis and how this affects the selection of data-driven methods is discussed in [7]. In [22], different open set recognition methods for fault diagnosis are discussed.

A common design principle in machine learning is trial and error, i.e. the engineer experimentally try to identify a good combination of features and data-driven models such that satisfactory performance is achieved. There is a need for data-driven methods using the ideas from model-based diagnosis to automate the design process. In e.g. [16], it is shown that redundancy is related to the manifold assumption, i.e. that observations from nominal system behavior are distributed on a low-dimensional manifold in observation space. The strong correlation between observations means that it is possible to predict one signal from the others which is needed to construct residual generators.

Measurement signals are often corrupted by some degree of noise which means that a consistency relation is almost satisfied in the nominal case or that observations are approximately distributed on a manifold. This means that some method is needed to evaluate if observations are likely to be distributed on a manifold. Several methods use Intrinsic Dimension (ID) [5] to estimate how many dimensions are sufficient to model the distribution of data, see e.g. [4]. In this work, intrinsic dimension is proposed to analyze redundancy properties from data and for data-driven design of diagnosis systems.

The outline of this paper is as follows. First, the problem formulation is presented in Section 2. The relationship between analytical redundancy and manifolds is discussed in Section 3. In Section 4, a data-driven diagnosis system design principle using intrinsic dimension is described. The proposed methods are evaluated in Section 5 using both a two-tank simulation model and data from a fuel injection system. The use of faulty data to design residuals that are insensitive to faults is illustrated in Section 6. Section 7 presents the conclusions from this work and proposed future works.

2 Problem formulation

The objective of this work is to investigate how redundancy can be evaluated directly from observations and how this information can be used in a data-driven diagnosis system design process. Specifically, the focus is on the relationship between redundancy and the intrinsic dimension of data, i.e., when the distribution of data can be approximated by a low-dimensional manifold in observation space. Then, a data-driven diagnosis system

design process, based on the principles of residual-based model-based design, is discussed and evaluated. Two case studies are considered: simulated data from a two-tank water tank system and real data from a fuel injection system in a heavy-duty diesel engine.

3 Estimating redundancy using intrinsic dimension

To formulate a redundancy property for data-driven diagnosis, the connection between analytical redundancy and manifolds is discussed. Then, it is described how the Intrinsic Dimension (ID) can be used to analyze the distribution of data including a brief summary of some methods used to estimate ID.

3.1 Analytical redundancy and manifolds

In residual-based diagnosis, residual generators are used to model the nominal relationship between known signals, e.g. actuators u and sensor signals y. This is done by first deriving consistency relations $0 = g(z)$, where $z = (u, y)$ from a model of the system. A consistency relation is, ideally, zero in the nominal case and deviates from zero when a fault occurs in the part of the system that is modeled by the relation [15]. The consistency relation is then used to construct a residual generator $r = g(z)$ to detect faults. A fault f_i is detectable if it is possible to construct a residual generator that is sensitive to the fault. A fault f_i is isolable from another fault f_j if it is possible to construct a residual generator that is sensitive to f_i but insensitive to f_j [15].

Different methods are used to derive consistency relations from a model, see e.g. [17]. For complex non-linear models, structural methods have been shown useful to analyze redundancy and find consistency relations. When constructing residual generators, several methods have focused on those using a minimal subset of observations, e.g. Minimally Structurally Over-determined (MSO) sets [17]. If one measurement is removed from the set, the remaining observations are no longer distributed on a low-dimensional manifold in the remaining observation subspace. Residuals constructed from MSO sets are good for fault isolation since they use a minimal set of observations. This might come at the cost of higher model inaccuracies [8].

The set of observations z that fulfill a set of consistency-relations is also called a solution manifold [23]. Thus, the set of possible observations z that can be explained by a fault-free system are modeled by the consistency relations. If the set of observations z is insensitive to a fault f_j then z will still belong to the solution manifold when the fault is present which is no longer true if the residual is sensitive to the fault [21].

A higher degree of redundancy means that more consistency relations can be constructed from the model. Assume that $g(z)$ is differentiable γ times. If $g(z) : \mathbb{R}^n \to \mathbb{R}^m$, and $\frac{\partial g}{\partial z}$ has full rank m, then the set of z that fulfill $0 = g(z)$ are distributed on a p-dimensional C^γ-manifold [23] where $p = n - m > 1$. Note that if derivatives of signals are needed to construct the consistency relations, then the derivatives are included as separate signals when defining z, see e.g. [16].

▶ **Example 1.** Consider a small system consisting of three sensors, y_1, y_2, and y_3, measuring the state x. The degree of redundancy is equal to 2 and an example of two consistency relations spanning all linear residual generators is $y_1 - y_2$ and $y_1 - y_3$. The solution manifold in the $n = 3$-dimensional observation space is given by $y_1 = y_2 = y_3$ which is a line, i.e. a 1-dimensional manifold. The number of known signals $n = 3$, consistency relations $m = 2$, and the dimension of the manifold $p = 1$ which is consistent with $p = n - m$.

For illustration, consider a linear model. The degree of redundancy of a linear model corresponds to how many linearly independent consistency relations that can be constructed, see e.g. [11]. Let

$$\dot{x} = Ax + Bu \tag{1}$$

where x is a state vector and A has full rank. Let A, and B be matrices of appropriate dimensions. For each value of u there exists values of x such that the equations are satisfied. Thus, the solution set $z = u$ is distributed in the whole observation space, which is spanned by u, i.e. the degree of redundancy is zero. Assume that sensors are added to the model as

$$y = x. \tag{2}$$

By adding sensors to the system, the dimension of the observation space n increases by one for each new signal. Also, the degree of redundancy increases by one for each new sensor, i.e. one more independent consistency relation can be constructed. The dimension of the solution manifold decreases by one with respect to n for each new consistency relation.

If a mathematical model of the system is not available, it is not possible to evaluate analytical redundancy. Instead, the idea here is to analyze when data is distributed on a lower-dimensional manifold to draw conclusions about the redundancy properties of the system. From a set of observations z, it is possible to select a subset of $\tilde{n}$ signals $\tilde{z} \subseteq z$ to formulate an observation space of dimension $\tilde{n}$. If the observations are distributed on a p-dimensional manifold then the degree of redundancy m can be computed as $m = \tilde{n} - p$. However, since signals are corrupted by noise, data dimension analysis methods are needed to estimate the dimension of the manifold. Here, the intrinsic dimension of observations will be used to approximate the dimension of the solution manifold.

3.2 Data dimension analysis

The ID essentially refers to the minimum number of variables needed to represent the underlying structure of a dataset without significant loss of information. Estimating the ID is crucial in various applications such as dimensionality reduction, data visualization, and understanding the complexity of datasets. Below is an outline of the ID estimation methods that are used in this work. The methods are implemented in the *scikit-dimension* python package, and were selected because of their favorable characteristics in the paper [4]. Moreover, After some initial testing, they were found to be the three most computationally efficient methods in this application, while still resulting in a good ID estimate.

Principal component analysis. Principal component analysis (PCA) is a commonly used linear projection method, that is searching for the best subspace projection that minimize the projection error. The main idea is to compute all eigenvalues of the data covariance matrix and based on the dominating eigenvalues, decide the ID. There exist multiple ID estimation methods that are based on PCA, e.g. methods that only return eigenvalues larger than a constant times the largest eigenvalue [12]. The authors of [6] have shown that many PCA estimation methods often result in poor estimations, especially for nonlinear and noisy data. However, there are more involved PCA based ID estimation methods that aims to improve the estimation for more complex data, such as [9].

Maximum likelihood estimation. The maximum likelihood estimation (MLE) is a statistical method used to estimate the ID of a dataset by maximizing the likelihood of the observed data under a given model. This approach focuses on the distribution of distances between neighboring points in the data to determine the ID.

One MLE-based algorithm for ID estimation was developed by Levina and Bickel [19]. This method starts by considering the k-nearest neighbors of each point in the dataset and for a given point, the distances to its nearest neighbors are used to estimate the local dimension around that point. The idea is that, in a space with a certain ID d, the distances to the nearest neighbors should follow a certain distribution related to d. An extension of this algorithm, proposed by Haro et al. [13], enhances the robustness of the original method, particularly in noisy environments.

Method of moments. Method of moments (MoM) is an estimation technique that derives the ID by using sample moments, such as mean and variance, with their theoretical counterparts. In the context of ID estimation, MoM involves using moments of distance distribution between data points. By matching the empirical moments calculated from the data with the expected moments under the assumption of a particular ID, it is possible to estimate the ID [3].

4 Design of data-driven residuals using intrinsic dimension

This section outlines a method for designing residuals based on the ID-based search of signal combinations. The process consists of four key steps: *data preprocessing*, *searching for low-dimensional signal combinations*, *selection of signal combinations*, and *design of residuals using the selected signal combinations*.

4.1 Data preprocessing

The first step is to determine which observations will be used for fault diagnosis and how it will be preprocessed. This preprocessing typically involves filtering to reduce noise, removing outliers, normalizing data, and potentially discarding corrupt or impractical data.

To model dynamic systems, derivatives are needed to form consistency relations. In many applications, the derivatives are not directly measured and must be numerically approximated. If the data is noisy or if higher order derivatives are needed, numerical approximations often results in poor-quality derivatives, which may complicate the evaluation of the ID. In this work, up to second-order derivatives of the observations have been estimated using a Savitzky-Golay filter [25]. The Savitzky-Golay filter is a non-causal smoothing filter that approximates the noise-free signal using low-degree polynomials. The set of observations z is constructed by combining actuator and sensor signals, and their higher-order derivatives.

4.2 Searching for low-dimensional signal combinations

Let $\tilde{z} \subseteq z$ denote a subset of signals such that $\tilde{z} \in \mathbb{R}^{\tilde{n}}$ where $\tilde{n} \leq n$. Based on the set of observations computed in the previous section, the next step involves identifying signal combinations $\tilde{z}$ where the ID is lower than the number of signals $\tilde{n}$ since that means that $\tilde{z}$ is approximately distributed on a low-dimensional manifold.

Since methods for evaluating ID are estimations, they do not provide a binary answer regarding whether the data distribution from a sensor combination is low-dimensional. The ID estimation methods described in Section 3.2 are local methods, i.e. the ID is evaluated by calculating the local ID for each sample using N nearest neighbors and then aggregating the information. The local ID distribution can also be used to calculate a global ID estimate, e.g. by taking the average of local ID distribution. However, taking the average value can be sensitive to outliers. Therefore, the distribution of the estimated local IDs is used to evaluate

the probability that the $\mathrm{ID}_{\tilde{z}}$, which denote the estimated ID for sensor combination $\tilde{z}$, is lower than $\tilde{n}$ minus an offset parameter α. If this probability exceeds a certain confidence threshold

$$\mathbb{P}(\mathrm{ID}_{\tilde{z}} < \tilde{n} - \alpha) \geq \theta, \tag{3}$$

the combination is considered low-dimensional, where θ is the required confidence level. Moreover, the number of nearest neighbors N must be determined. Together, these three parameters, α, θ, and N, affect the search results. The probability in (3) is estimated using a kernel density approximation (KDE) to approximate the distribution [14].

The objective is then to find all signal combinations $\tilde{z}$ such that $\mathrm{ID}_{\tilde{z}} < \tilde{n} - 1$, i.e. if (3) is satisfied. Note that if nominal data of $\tilde{n}$ is distributed on a low-dimensional manifold then all supersets of $\bar{z}$ are also distributed on low-dimensional manifolds. Thus, it is sufficient to find minimal signal combinations such that (3) is satisfied, i.e. when $\mathrm{ID}_{\tilde{z}} = \tilde{n} - 1$, instead of evaluating all signal combinations. Finding the minimal combinations is a combinatorial problem and the search space grows exponentially with n. Another important factor is the curse of dimensionality, i.e., when analyzing data in higher dimensions, the number of samples needed in training data to estimate the data distribution grows exponentially with $\tilde{n}$, see e.g. [28]. Therefore, the combinations $\tilde{z} \subseteq z$ are evaluated using a breadth-fist search principle, starting with the smallest combinations. This ensures that the minimal combinations are found first, and the search is terminated when no more minimal signal combinations can be identified.

4.3 Selection of signal combinations

After all minimal observation combinations have been found, the next step is to select a subset of these combinations for residual design. The sensor combination $\tilde{z}$ can be evaluated using different criteria. Here, each $\tilde{z}$ is used to fit a regression model using one of the signal as a target signal and evaluate the prediction error. For each $\tilde{z}$, all signals are evaluated as target signals predicted using a regression model based on the other signals in the set and the best prediction accuracy, quantified using the normalized root mean squared error (NRMSE), is selected as a metric. Here, linear regression models are used to reduce computation time when evaluating many combinations. The combinations which has a low NRMSE value and $\tilde{n} - \mathrm{ID}_{\tilde{z}} \approx 1$ are then selected for residual generation.

4.4 Residual design

When a number of minimal sensor combinations has been identified, the last step is to design residual generators based on the selected combinations. The primary idea is to use a data-driven approach to predict one the signals using the other within each selected combination, and by doing this, constructing a residual based on the prediction error. This can be achieved through various methods, such as linear regression models, neural networks, or other techniques, depending on the system's complexity.

The objective of this work is illustrating the potential of using ID. Therefore, standard linear regression modeling techniques will be used in the case studies, see e.g. [20].

5 Evaluation of method for data-driven residual design

This section will present the result of using the residual design method described in Section 4. The proposed design method for data-driven residuals is evaluated first using a simulation model of a non-linear two-tank system to compare the results with a structural analysis

of the simulation model. Then, data from a fuel injection system in a heavy-duty truck is used to validate the method. The maximum likelihood estimation method was used when evaluating the ID for both systems.

5.1 Water tank system

The water tank system consists of two connected tanks, as illustrated in Figure 1 where the upper tank is filled by a controlled pump, and the water flows from the upper to the lower tank. The model for this system, derived from first principles, is described by the set of equations:

$$\begin{aligned} \dot{h}_1 &= d_1 u - d_2\sqrt{h_1} + f_a, & y_1 &= h_1 + e_1, \\ \dot{h}_2 &= (1-f_l)d_3\sqrt{h_1} - (1-f_c)d_4\sqrt{h_2}, & y_2 &= h_2 + e_2, \\ & & y_3 &= d_5\sqrt{h_1} + e_3, \\ & & y_4 &= (1-f_c)d_6\sqrt{h_2} + e_4 \end{aligned} \tag{4}$$

where h_1, h_2 are the water levels in the upper and lower tanks, respectively, u is the control signal to the pump, y_i are the measurement signals, d_i are model parameters, e_i are measurement noise, here modeled as Gaussian distributed, f_a is a fault in the pump actuator, f_l is a leakage between the upper tank and sensor y_3, and f_c is a clogging in the output of the lower tank.

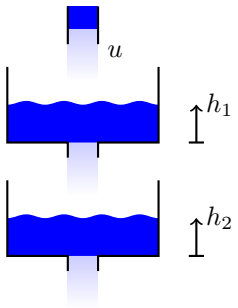


Figure 1 An illustration of the two tank system.

A structural model is derived based on the model equations (4) and implemented in Fault Diagnosis Toolbox [10]. The set of MSO candidates based on the model are identified for comparison with the identified observation combinations identified from data using ID.

5.1.1 Data description

Seven datasets are generated from the simulation model of the water tank system, including fault-free and faulty scenarios. All dataset include the following set of signals:

$$z = \{y_1, y_2, y_3, y_4, \dot{y}_1, \dot{y}_2, \dot{y}_3, \dot{y}_4, \ddot{y}_1\, \ddot{y}_2\}.$$

For simplicity, the set of signals were selected based on the model structure. Since the model had two dynamic equations, the first and second derivatives of the signals were included. Note that higher derivatives of the signals could be included, but the number of signals was intentionally kept low to reduce the number of combinations. Since only the values y_i are directly obtained, the first- and second-order time derivatives are numerically approximated. In this case, they are approximated using a Savitzky–Golay filter [25].

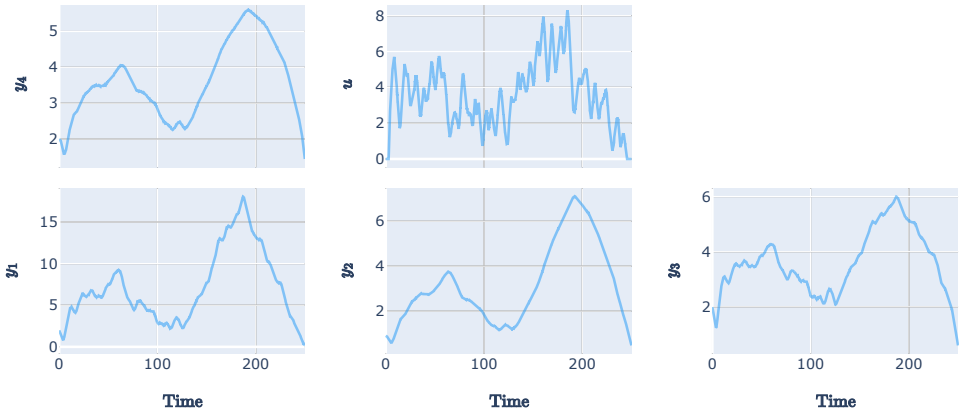


Figure 2 Example of measurements from the simulated water tank system.

The first four datasets represent nominal operation data: two are used for evaluating the ID and selecting combinations for residual design. The third and fourth datasets are used for testing the residuals, with one dataset for training the model and the other for evaluation. The remaining three datasets correspond to three different fault scenarios: a fault in the actuator (f_a), leakage between the upper tank and sensor 3 (f_l), and clogging in output of lower tank (f_c). These datasets are used to evaluate how the designed residuals react to the different faults. Figure 2 shows the signals from one of the nominal datasets, excluding the derivatives.

5.1.2 Intrinsic dimension evaluation

In addition to evaluating the ID for all signal combinations, the Normalized Root Mean Squared Error (NRMSE) of the prediction error is calculated after fitting a linear regression model to the data. Since the water tank system is non-linear, the linear regression model was extended to include the squares and square roots of all signals based on knowledge of the system behavior.

Figure 3 shows the relationship between evaluated ID and NRMSE for all 4083 signal combinations. The NRMSE was normalized by the standard deviation of the target signal to allow for a fair comparison between the different regression models predicting different target signals. The x-axis shows the difference between the number of signals in the set (the color of each point) and the estimated ID from the data. Observation sets where the dimensional difference is large result in regression models utilizing many signals for prediction which gives a better overall prediction accuracy. For dimensional difference one and higher, there are sensor combinations that achieve close to perfect prediction accuracy. For a dimensional difference smaller than one, the best NRMSE increases which indicates that there is insufficient information in the regression models to predict the target signals. This observation agrees with the expected behavior when observations are not distributed on a low-dimensional manifold.

The MSO sets derived from the water tank model (4), represent the minimal equation sets to construct consistency relations. Based on the MSO sets it is also possible to derive the minimal sets of observations needed to construct consistency relations based on the model. The sensor combinations that correspond to the MSO sets derived from the model are highlighted as stars in Figure 3. Ideally, the dimensional difference of the MSO sets should be close to one, and the NRMSE should be small. As shown in the figure, while not perfect, the MSO sets generally follow this expectation.

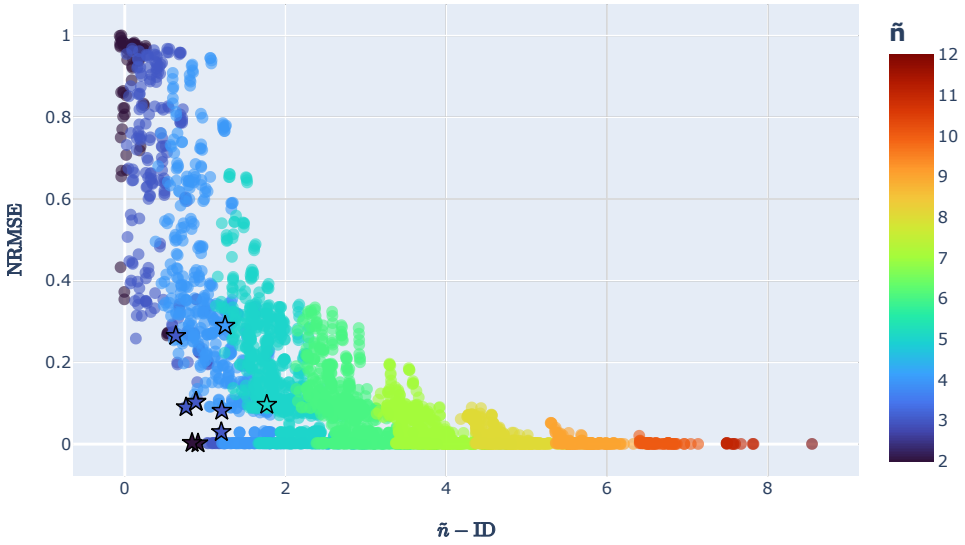


Figure 3 Normalized root mean squared error versus dimensional difference for all signal combinations for the water tank system. The stars represent the sensor combinations corresponding to MSO sets derived from the model (4). Marker color represent combination dimension, i.e. the number of signals in the set.

After evaluating all minimal signal combinations, as described in Section 4.2, 209 combinations were identified as minimal. The top six ranked combinations, those with the highest probability and a NRMSE of less than 0.2, were selected as candidates for residual design.

The parameters used for evaluating all combinations and finding minimal combinations were selected as $N = 70$, $\alpha = 0.5$, and $\theta = 0.70$. These values were identified experimentally based on their impact on the resulting minimal combinations. Both parameters α and θ influence the strictness of the search process similarly: increasing their values makes the search criteria more strict, reducing the likelihood of identifying weaker signal combinations. There is a trade-off between the risk of overlooking important combinations with higher parameter values and the risk of detecting false positives with lower values. As mentioned in [13] increasing the number of neighbors reduces the ID estimation bias when using the MLE method. However, there is a compromise, since increasing N might affect underlying assumptions of constant density in the neighborhood, which might affect the results negatively. This means that tuning of N is required that depends on the properties of the data.

The process for designing residuals using these combinations will be described next.

5.1.3 Residual design and evaluation

The residual generators are constructed based on the six selected sensor combinations using the same linear regression models as to compute the NRMSE.

Figures 4-6 show the performance of the residuals when subjected to the three modeled faults: actuator fault (f_a), clogging (f_c), and leakage (f_l), respectively. In these figures, the blue line is the residual under nominal conditions, while the orange line is the residual affected by the fault. Any overlap between the two lines is indicated by a green color. All three faults are introduced in the system using similar fault scenarios. Each fault signal behaves as a ramp starting at $t = 10$ second that increases to an amplitude of 0.4 over 2

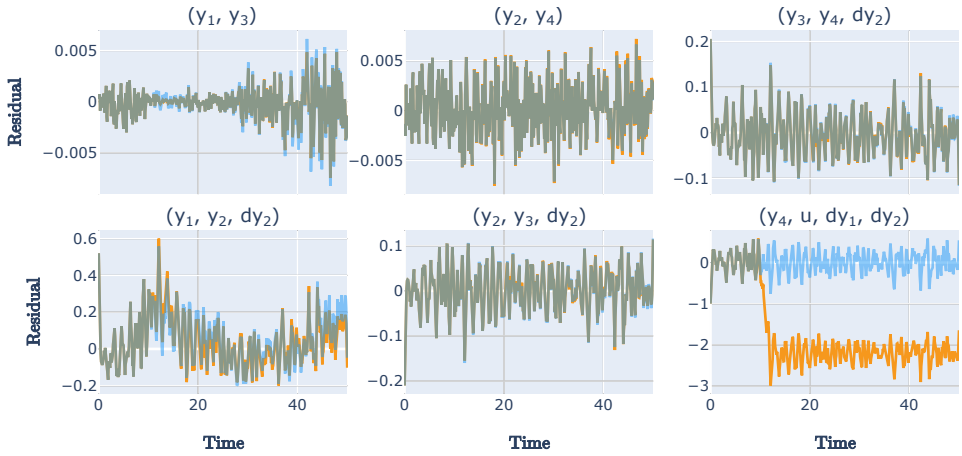


Figure 4 Comparison of residuals between fault-free data and data under an actuator fault, which occurs at $t = 10$ s.

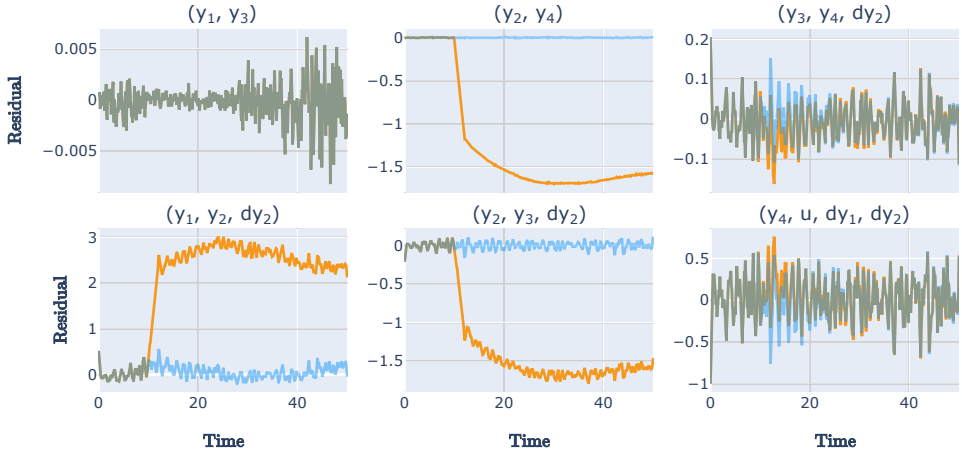


Figure 5 Comparison of residuals between fault-free data and data under a clogging fault, which occurs at $t = 10$ s.

seconds. The results demonstrate that the residuals maintain a low value under nominal conditions, while different residuals reacts to the different faults, indicating their potential for fault isolation.

It is also possible to analyze which part of the system that is modeled in each residual generator based on the selected the sensor combinations. For example the residual generator based on the signals y_1, y_2, and $\dot{y}_2$ is modeling the relationship between signals related to the dynamics of tank two based on the water level of tank one, see (4). This is also visible in the residual plots since the residual reacts to a clogging in tank one in Figure 5, which affects how much water is flowing into tank two, and a leakage in tank two in Figure 6, which affects the behavior the tank. However, the actuator fault in Figure 4 is only affecting how much water that flows into tank one but not the relationship between y_1, y_2, and $\dot{y}_2$. This shows that it is possible reason about the cause of triggered data-driven residuals based on physical insights about the system.

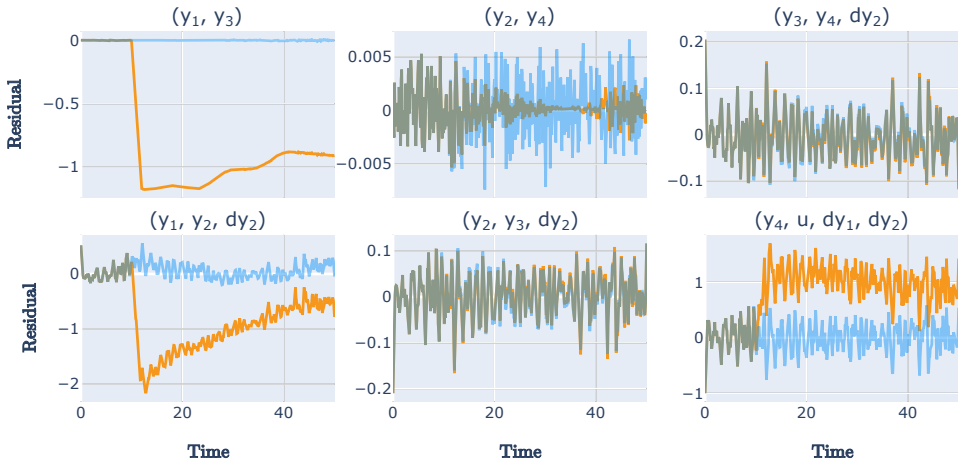


Figure 6 Comparison of residuals between fault-free data and data under a leak fault, which occurs at $t = 10$ s.

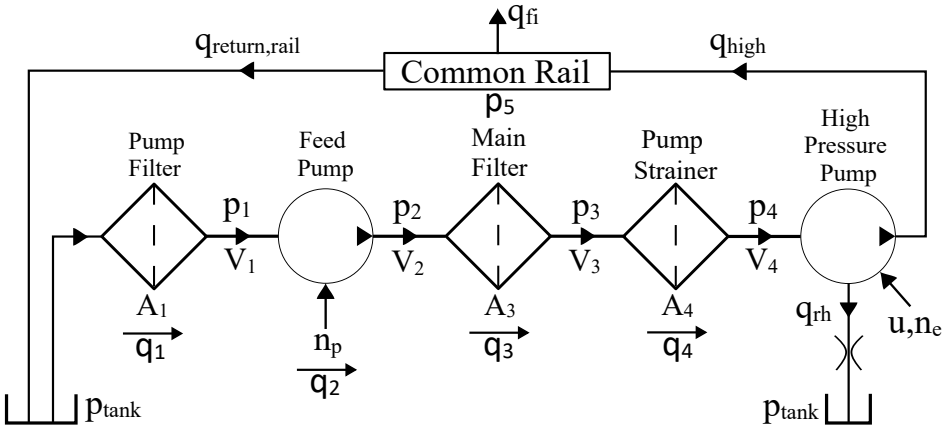


Figure 7 A schematic of the fuel injection system. The figure is used with permission from [2].

5.2 Fuel injection in heavy-duty diesel engine

The second case study is a fuel injection system in a heavy-duty diesel engine. The same case study has previously been analyzed in [2]. The system consists of two pumps used to provide fuel that is injected into the cylinders in a diesel engine. A schematic of the system is shown in Figure 7 where fuel from the tank is pumped through a set of filters and finally to the common rail where it is injected to the cylinders.

The amount of fuel injected into the cylinders is not measured but is controlled by adjusting how long a set of solenoid valves are open. Thus, the system acts to maintain a fixed pressure in the common rail to inject the correct amount of fuel. The available signals from the fuel injection system are:

- Pump speeds y_{np} and y_{ne},
- LP and the HP circuit pressures y_{p2} and y_{p5},
- HP pump solenoids duty cycle u,
- Estimated injected fuel to the cylinders $y_{q,fi}$.

5.3 Data description

Based on the results in [2], it was observed that the dynamics could be approximated by a second order system. Therefore, the extracted set of observations includes the following signals:

$$z = \{y_{np}, y_{ne}, y_{p2}, y_{p5}, y_{q,fi}, u, \dot{y}_{ne}, \dot{y}_{p5}, \dot{y}_{q,fi}, \dot{u}, \ddot{y}_{p5}, \ddot{y}_{q,fi}\},$$

where the derivatives of the signals are numerically approximated, similar as for the water tank system.

The fuel injection system data consists of four datasets, all collected during real-world operation of a heavy-duty truck:

- Fault-free data from 40-tonne load (30 min)
- Fault-free data from no load (20 min)
- Clogging in the main filter (28 min)
- Degradation of the HP pump (4 min)

The first two datasets represent nominal operation: one with the truck under a 40-ton load and the other with no load. The dataset recorded under load conditions is used for performing the ID evaluation and designing the residuals, while the no-load dataset is used to evaluate these residuals. The remaining two datasets were recorded under faulty conditions: one with a clogged main filter in the fuel injection system, and the other with a pump-related failure. These datasets are used to assess how the faults impact the designed residuals. Figure 8 shows the data from the 40-tonne load nominal dataset.

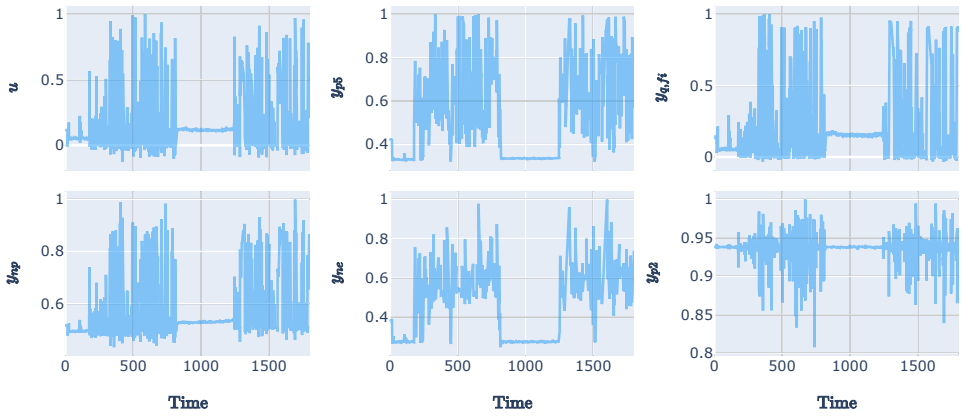


Figure 8 Example of measurements from the fuel injection system in a heavy-duty truck.

The evaluation procedure follows the same methodology used for the water tank system described in Section 5.1. However, there is no comparison of the solution to any MSO sets.

5.3.1 Intrinsic dimension evaluation

The 12 selected signals in z result in 4083 sensor combinations in total. Out of all of these combinations, 462 were classified as low-dimensional with the parameters: $\alpha = 0.5$, $\theta = 0.65$, and $N = 160$. These parameters were based on those used in the water tank system, but were adjusted to account for the larger and noisier dataset. Specifically, the number of neighbors was increased to improve the results, and the search strictness was slightly reduced by lowering θ to avoid overlooking potentially relevant combinations.

Here, data is used to fit a linear regression model to compute the prediction error. Based on the results in [2], it was concluded that a linear model was sufficient to model the system behavior. Figure 9 shows the relationship between NRMSE and dimensional difference for all combinations. The results show the same characteristic behavior as for the water tank system where a higher dimensional difference result in a lower prediction error. Still, the objective is to design residual generators using minimal sensor sets. Analogously to Section 5.1, six candidates were selected to be used for residual design. Because of the higher RMSE in this case, the candidates were chosen to be below a NRMSE of 0.4 instead of 0.2.

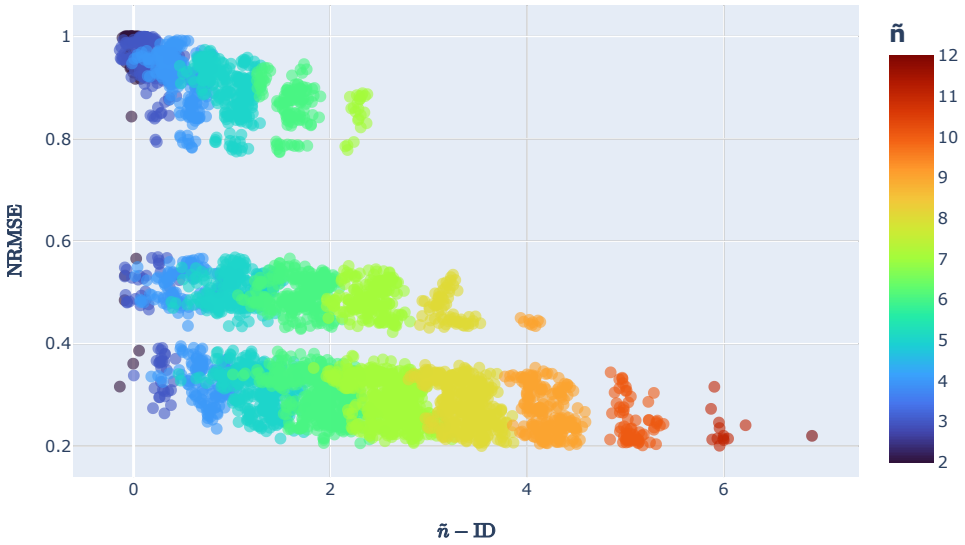


Figure 9 Normalized root mean squared error versus dimensional difference for all signal combinations for the fuel injection system. Marker color represent the dimension of the observation space, i.e. the number of signals $\tilde{n}$.

5.3.2 Residual design and evaluation

Figures 10 and 11 show how the residuals react to the two different faults, both time series data and the distributions of the residual during nominal (blue) and faulty conditions (orange). Even tough the residuals are affected by noise, the faults are detectable. Since each fault is resulting in a bias in the residual output, the noise can be suppressed using a low-pass filter without significantly reducing detection performance. The results from the analysis of the fuel injection system show that the proposed design method can be used for designing data-driven residual generators for fault diagnosis in industrial applications.

6 Identifying sensor combinations for fault isolation using augmented training data

The residual design process for the two case studies in the previous section identified sensor combinations based on the prediction error. This means that there is no information about the fault sensitivity of the constructed residual generators. In [21], it is shown that data from decoupled faults are distributed along the same manifold as fault-free data. Thus, augmenting the training data used in the ID evaluation with data from faults that should be decoupled could be used to find signal combinations that are insensitive to the fault.

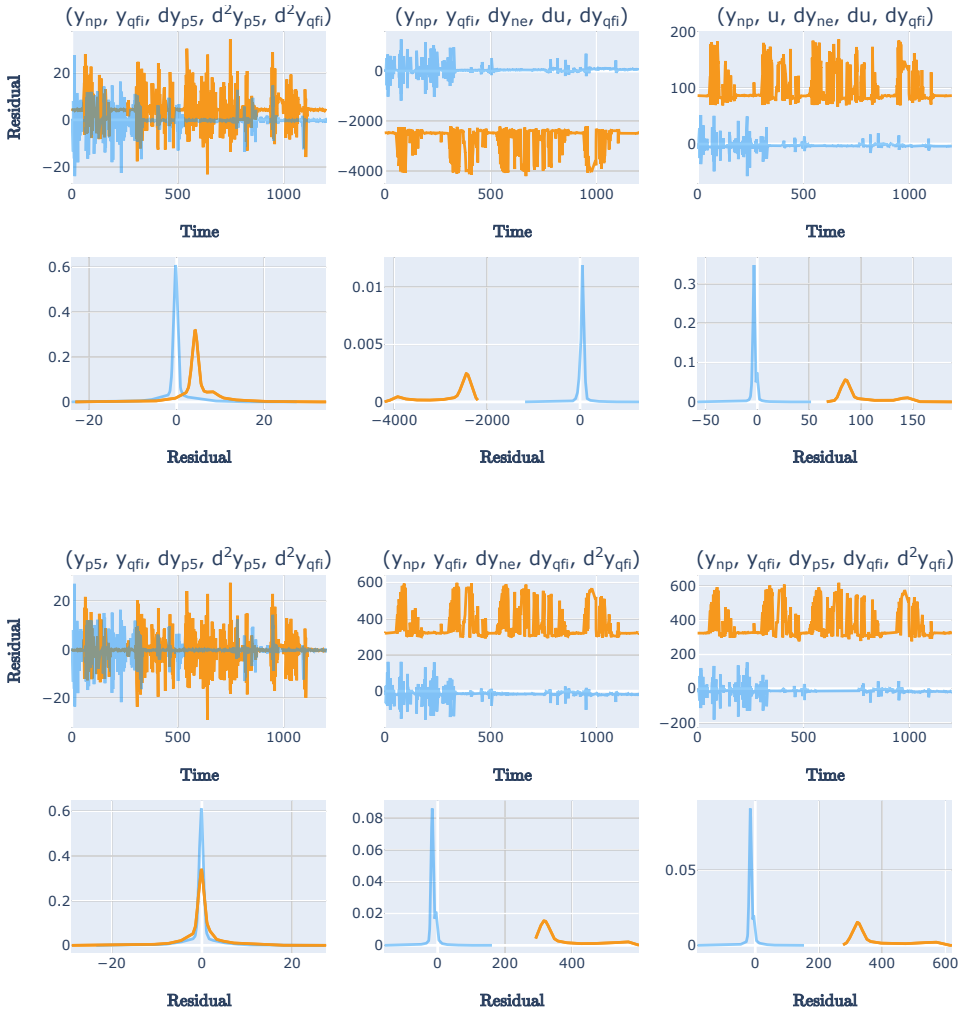


Figure 10 Residual comparison between no fault and clogging in the main filter. Each residual have two separate plots, the upper plot showing the time-series data of the residuals, and the lower plot showing the residual distribution.

The design process in the previous section for the water tank system using training data augmented with data from the clogging fault. A new set of sensor combinations is identified and used to construct residual generators. Figure 12 shows the new selected residuals based on the six top ranked signal combinations, when performing exactly the same procedure as previously. All residuals are insensitive to the clogging fault, demonstrating the potential of finding signal combinations that can be used to form residuals unaffected by a chosen fault. This is promising, since have residuals that are affected by different faults is essential for consistency-based fault isolation.

Experiments also showed that for the ID algorithm to identify sensor combinations that were insensitive to the fault in the augmented training data required sufficient excitation of the fault. For example, a constant fault that is only introducing a shift in the measurements will not affect the estimated ID since the local ID around those measurements will be the same.

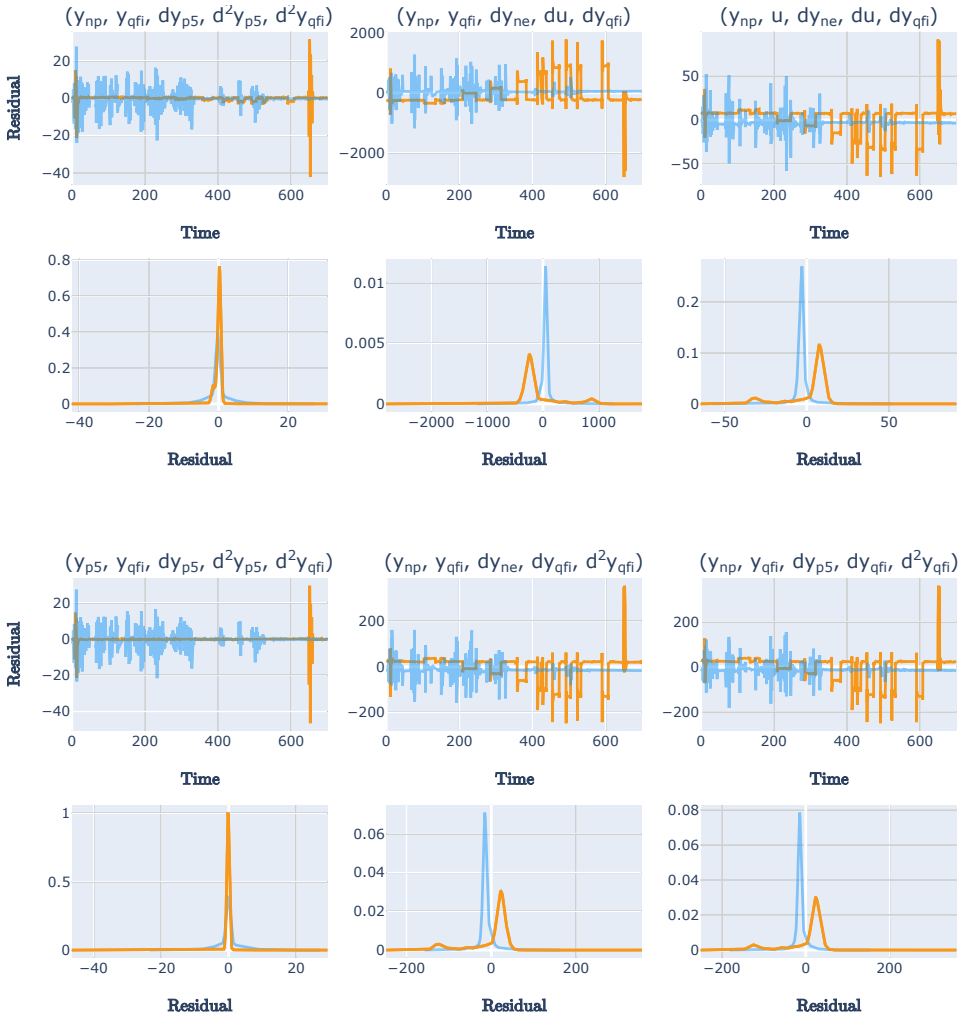


Figure 11 Residual comparison between no fault and a pump related fault.

7 Conclusions and future works

Even though data-driven diagnosis is often treated as a standard classification problem, it is shown that theory from model-based diagnosis, e.g. redundancy, can be used in a data-driven context. When representative training data is limited, there is a risk of misclassifications due to generalization problems. Thus, theory and methods for data-driven fault diagnosis are needed. The connection between analytical redundancy and ID is evaluated using data from a simulated two-tank system. The results show the connection between model-based methods to find consistency relations and the distribution of data. The results when evaluating the prediction performance of different sensor combinations validate the fact that prediction accuracy is improved by utilizing more observations from the system. However, from a fault isolation perspective it is more relevant to identify minimal subsets of observations for residual generation, similar to MSO sets in model-based diagnosis. A data-driven diagnosis system design process is also proposed using the ID to find candidate observation sets for residual generation. The proposed method is validated using both the simulation model of the two tank system and data from a fuel injection system.

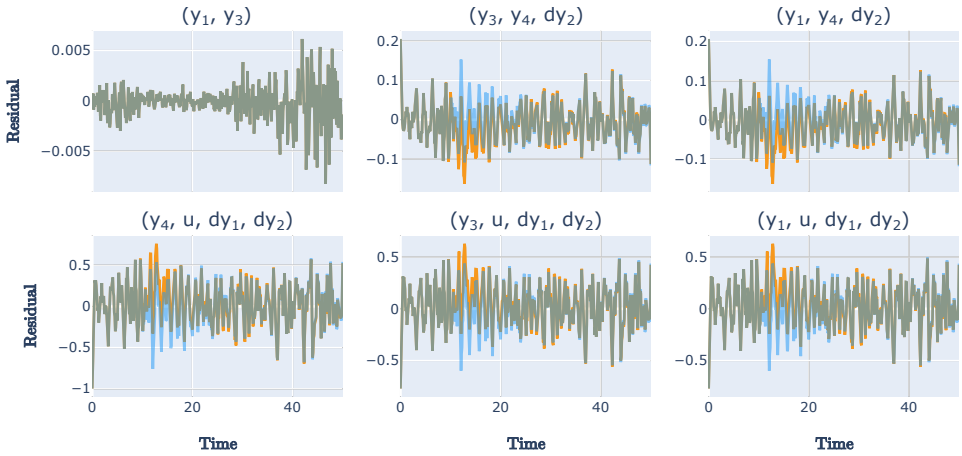


Figure 12 Comparison of residuals between normal data and data under a clogging fault, which occurs at $t = 10$ s.

As future work, more investigations are needed to efficiently estimate the ID using noisy data. Here, derivative causality was considered, i.e. signals and their derivatives, which has limitations for complex dynamic systems. Thus, it should be investigated how to apply ID for integral causality, e.g. how to directly analyze time-series data instead of estimating the derivatives.

References

1 A. Abid, M. Khan, and J. Iqbal. A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review*, 54:3639–3664, 2021. doi:10.1007/S10462-020-09934-2.

2 N. Allansson, A. Mohammadi, D. Jung, and M. Krysander. Fuel injection fault diagnosis using structural analysis and data-driven residuals. *IFAC-PapersOnLine*, 58(4):360–365, 2024.

3 L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. Houle, K. Kawarabayashi, and M. Nett. Extreme-value-theoretic estimation of local intrinsic dimensionality. *Data Mining and Knowledge Discovery*, 32(6):1768–1805, 2018. doi:10.1007/S10618-018-0578-6.

4 J. Bac, E. Mirkes, A. Gorban, I. Tyukin, and A. Zinovyev. Scikit-dimension: a python package for intrinsic dimension estimation. *Entropy*, 23(10):1368, 2021. doi:10.3390/E23101368.

5 R. Bennett. The intrinsic dimensionality of signal collections. *IEEE Transactions on Information Theory*, 15(5):517–525, 1969. doi:10.1109/TIT.1969.1054365.

6 F. Camastra and A. Staiano. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328:26–41, 2016. doi:10.1016/J.INS.2015.08.029.

7 X. Dai and Z. Gao. From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis. *IEEE Transactions on Industrial Informatics*, 9(4):2226–2238, 2013. doi:10.1109/TII.2013.2243743.

8 D. Eriksson, E. Frisk, and M. Krysander. A method for quantitative fault diagnosability analysis of stochastic linear descriptor models. *Automatica*, 49(6):1591–1600, 2013. doi:10.1016/J.AUTOMATICA.2013.02.045.

9 M. Fan, N. Gu, H. Qiao, and B. Zhang. Intrinsic dimension estimation of data by principal component analysis. arxiv. *Preprint]. doi*, 10, 2010.

10 E. Frisk, M. Krysander, and D. Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293, 2017.

11 E. Frisk and M. Nyberg. A minimal polynomial basis solution to residual generation for fault diagnosis in linear systems. *Automatica*, 37(9):1417–1424, 2001. doi:10.1016/S0005-1098(01)00078-4.

12 K. Fukunaga and D. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on computers*, 100(2):176–183, 1971. doi:10.1109/T-C.1971.223208.

13 G. Haro, G. Randall, and G. Sapiro. Translated poisson mixture model for stratification learning. *International Journal of Computer Vision*, 80:358–374, 2008. doi:10.1007/S11263-008-0144-6.

14 T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: data mining, inference, and prediction, 2017.

15 D. Jung. Isolation and localization of unknown faults using neural network-based residuals. In *Annual Conference of the PHM Society*, volume 11, 2019.

16 D. Jung, M. Krysander, and A. Mohammadi. Fault diagnosis using data-driven residuals for anomaly classification with incomplete training data. *IFAC-PapersOnLine*, 56(2):2903–2908, 2023.

17 M. Krysander, J. Åslund, and M. Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):197–206, 2007. doi:10.1109/TSMCA.2007.909555.

18 M. Krysander and E. Frisk. Sensor placement for fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(6):1398–1410, 2008. doi:10.1109/TSMCA.2008.2003968.

19 E. Levina and P. Bickel. Maximum likelihood estimation of intrinsic dimension. *Advances in neural information processing systems*, 17, 2004.

20 L. Ljung. *System identification (2nd ed.): theory for the user*. Prentice Hall PTR, USA, 1999.

21 A. Mohammadi, M. Krysander, and D. Jung. Analysis of grey-box neural network-based residuals for consistency-based fault diagnosis. *IFAC-PapersOnLine*, 55(6):1–6, 2022.

22 A. Rehman, W. Jiao, J. Sun, H. Pan, and T. Yan. Open set recognition methods for fault diagnosis: A review. In *2023 15th International Conference on Advanced Computational Intelligence (ICACI)*, pages 1–8. IEEE, 2023.

23 W. Rheinboldt. On the computation of multi-dimensional solution manifolds of parametrized equations. *Numerische Mathematik*, 53(1):165–181, 1988.

24 C. Sankavaram, A. Kodali, K. Pattipati, and S. Singh. Incremental classifiers for data-driven fault diagnosis applied to automotive systems. *IEEE access*, 3:407–419, 2015. doi:10.1109/ACCESS.2015.2422833.

25 A. Savitzky and M. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

26 A. Theissler, J. Pérez-Velázquez, M. Kettelgerdes, and G. Elger. Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability engineering & system safety*, 215:107864, 2021. doi:10.1016/J.RESS.2021.107864.

27 L. Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27:1–16, 2014. doi:10.1016/J.ENGAPPAI.2013.09.018.

28 M. Verleysen and D. François. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

29 Y. Xu, S. Kohtz, J. Boakye, P. Gardoni, and P. Wang. Physics-informed machine learning for reliability and systems safety applications: State of the art and challenges. *Reliability Engineering & System Safety*, 230:108900, 2023. doi:10.1016/J.RESS.2022.108900.

30 Z. Xu and J. Saleh. Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. *Reliability Engineering & System Safety*, 211:107530, 2021. doi:10.1016/J.RESS.2021.107530.

31 T. Zhang, J. Chen, F. Li, K. Zhang, H. Lv, S. He, and E. Xu. Intelligent fault diagnosis of machines with small & imbalanced data: A state-of-the-art review and possible extensions. *ISA transactions*, 119:152–171, 2022.

Bridging Hardware and Software Diagnosis: Leveraging Fault Signature Matrix and Spectrum-Based Fault Localization Similarities

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Franz Wotawa
Christian-Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems, Institute of Software Technology, Graz University of Technology, Austria

Abstract

This paper examines two prominent Fault Detection and Isolation methodologies: the Signature Matrix approach, traditionally used in hardware systems, and the Spectrum-based approach, applied in software fault localization. Despite their distinct operational domains, both methods share the objective of precisely identifying and isolating faults. This study aims to compare these approaches and to highlight their similarities in principle. Through a comparative analysis, we assess how the structured pattern recognition of the Signature Matrix method and the statistical analysis capabilities of the Spectrum-based approach can be synergized to enhance diagnostic processes of cyber-physical systems that are composed of both hardware and software components. The investigation is motivated by the prospect of developing a hybrid Fault Detection and Isolation strategy that incorporates the robust detection mechanisms of hardware diagnostics with the techniques used in software fault localization. The findings are intended to advance the theoretical framework of Fault Detection and Isolation systems and suggest practical implementations across varied technological platforms, thereby improving the reliability and efficiency of fault detection and isolation in both hardware and software contexts.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Software and its engineering → Software defect analysis; Hardware → Error detection and error correction

Keywords and phrases Diagnosis, Fault detection and identification, Software debugging

Digital Object Identifier 10.4230/OASIcs.DX.2024.5

Funding *Louise Travé-Massuyès*: This work is partially supported by 3IA Artificial and Natural Intelligence Toulouse Institute (ANITI), French "Investing for the Future - PIA3" program under the Grant agreement ANR-19-PI3A-0004".
Franz Wotawa: The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged.

1 Introduction

In the field of Fault Detection and Isolation (FDI), innovative methodologies have significantly advanced the diagnosis and management of faults in both hardware and software systems. Among these, the Signature Matrix approach and the Spectrum-based approach, particularly as applied in software fault localization, stand out as critical tools in the arsenal of diagnostic techniques. The Signature Matrix method relies on a well-defined structure where fault signatures are cataloged in a matrix format, facilitating the pinpointing of system faults and anomaly detection. On the other hand, the Spectrum-based approach, widely utilized in software fault localization, leverages the statistical analysis of pass/fail data across various execution points to identify potential faults.


© Louise Travé-Massuyès and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



Despite their application to different domains – hardware and software respectively – both approaches share a foundational goal: to systematically identify and isolate faults with high accuracy. This common objective underscores the importance of comparing these methodologies to explore their potential synergies and complementary capabilities. The Signature Matrix approach, with its robust framework for hardware systems, and the Spectrum-based approach, with its nuanced analysis of software execution, both strive to enhance fault diagnostic processes. They do so along paths that appear different at first glance but have much in common.

The motivation for this comparative study arises from the potential to cross-pollinate between these domains, leveraging insights from one to enrich the other. Such a comparison not only elucidates the strengths and weaknesses inherent in each method but also sets the stage for hybrid diagnostic strategies that could incorporate the precision of signature matrices and the statistical analysis capabilities of spectrum-based methods. This paper aims to bridge the methodological divide between hardware fault detection and software fault localization, proposing avenues for integrated approaches that enhance the reliability and efficiency of FDI systems across different technological realms. Through a detailed comparative analysis, we seek to contribute to both the theoretical advancement and practical application of FDI methodologies in complex engineering and software systems.

For this purpose, we briefly introduce the basic foundations of both approaches utilizing simple examples for illustration purposes. Furthermore, we discuss the similarities and differences between signature matrix approach and spectrum-based fault localization, and suggest a combined approach focusing on the integration of methods. The latter may lead to a methodology that can be handle diagnosis of hardware and software altogether.

2 The Fault Signature Matrix approach

The FDI control community focuses predominantly on physical systems. These systems, which include mechanical, electrical, and hydraulic components, among others, are integral to industries ranging from manufacturing to aerospace. The models used to represent these systems are typically derived from physical laws and principles, ensuring that the simulations and diagnostics closely mirror real-world dynamics. Such models are crucial for accurately predicting system behaviors and effectively diagnosing issues, providing a reliable basis for further analysis and application in practical settings. The emphasis on physical systems and their corresponding models underscores the importance of precise and applicable solutions in the domain of system engineering.

Diagnosis techniques are based on behavioral models that define the relationships between system quantities. The typical model may be formulated in the temporal domain, then known as a *state-space model* :

$$\Sigma(z,x): \begin{cases} dx(t)/dt = h(x(t),u(t),\theta), \\ y(t) = g(x(t),u(t),\theta), \\ x(t_0) = x_0, \end{cases} \tag{1}$$

where the functions h and g are linear or nonlinear functions. $x(t) \in \mathbb{R}^n$ denotes the state variable vector, whose components are the internal variables to the system that cannot be directly measured. These variables form the set of unknown variables X. $y(t) \in \mathbb{R}^m$ and and $u(t) \in \mathbb{R}^l$ denote the output and input variable vectors, respectively. Output and input variables are measured. They respectively stand for the measurements delivered by sensors and for the controls applied by actuators to the system. They are gathered in the vector $z(t)$

and they form the set of known variables Z, also called *observations*. $u(t)$ may be equal to 0 in case of an uncontrolled system. The variables z(t) and x(t) are time-dependent functions. It is assumed that all sensor and control values are acquired/applied synchronously according to a given sampling rate. Therefore the explicit mention to time may be omitted and we may write z and x instead of $z(t)$ and $x(t)$, respectively.

In a more abstract form, a system model can be defined as follows.

▶ **Definition 1** (System model). *A system model $\Sigma(z, x)$ is composed of a set of algebraic and/or differential equations $e_k(z^k, x^k), k = 1, \ldots, n_e$, where z^k is a subvector of the vector of known variables z, and x^k is a subvector of the vector of unknown variables x.*

At the heart of FDI control methods lies the concept of *residual*, and a key challenge is the *generation of residuals*. Residual generators define *diagnosis tests*. The evaluation of residuals indeed brings the information of consistency or non consistency of a model with observations.

▶ **Definition 2** (Consistency of observations with a model). *An observed trajectory z is deemed consistent with a model $\Sigma(z, x)$ if there is a trajectory for x that satisfies the equations of $\Sigma(z, x)$.*

▶ **Definition 3** (Diagnosis test for $\Sigma(z, x)$). *A diagnosis test $\mathcal{T}$, also called a residual generator, for the model $\Sigma(z, x)$ is a system that inputs a subset of known variables $\tilde{Z} \subseteq Z$ and their derivatives and outputs a scalar r, named the* residual *such that, for all z consistent with $\Sigma(z, x)$, $\lim_{t\to\infty} r(t) = 0$.*

When the model aligns with observations, the residuals tend to zero as time t tends to infinity. Conversely, discrepancies in residuals may occur if the model does not align. Practically, residuals are never precisely *zero* due to the impracticality of the noise-free assumption inherent to system (1). The evaluation of residuals involves statistical testing that considers the noise characteristics, as discussed in sources like [1]. Furthermore, optimization of residuals aims to enhance robustness against disturbances [7] and to incorporate uncertainties [4, 8]. In this paper, we assume an ideal system without noise.

Diagnosis tests are derived from the model (1) by using different subsets of equations, each forming a minimal structurally overdetermined (MSO) set [6], i.e. a subset of equations just overdetermined. A diagnosis test $\mathcal{T}$ may take the form of an Analytical Redundancy Relation (ARR) in which case $\mathcal{T}$ expresses as a relation of the form $r = arr(\bar{z}')$, where r is the residual and $\bar{z}'$ is a subvector of $\bar{z}$, and $\bar{z}$ stands for z and its time derivatives up to some (unspecified) order.

Assumption 1 – Let us assume, without loss of generality, that every equation of $\Sigma(z, x)$ represents the behavior of one and only one component $\mathcal{C} \in COMPS$, where $COMPS$ is the set of components composing the system $\Sigma(z, x)$.

▶ **Definition 4** (Equation/component support of a diagnosis test). *The equation/component support $\Sigma' \subseteq \Sigma$ of a diagnosis test $\mathcal{T}$ is defined as the set of equations of the MSO set underlying $\mathcal{T}$, i.e., the subset of equations of model (1) that are used to derive $\mathcal{T}$. Given Assumption 1, the component support of a diagnosis test is isomorphic to its equation support.*

In general, two categories of faults are recognized: additive faults and multiplicative faults. Additive faults typically introduce a bias into the system, altering measurements or actuators by a consistent amount. This can occur due to issues like incorrect settings on an analog-to-digital converter or erroneous application of control voltages. On the other hand, multiplicative faults, also referred to as parameter faults, impact the system's dynamics and can compromise the stability of the system and restrict the functionality of components.

▶ **Definition 5** (Additive Fault). *An additive fault is characterized by the alteration of a measured quantity $z_i \in \mathbb{R}$ through the addition of an offset $f_i \in \mathbb{R}$, resulting in the sensor/actuator reading $z_i' = z_i + f_i$, rather than the original z_i.*

▶ **Definition 6** (Multiplicative Fault). *A multiplicative fault involves the alteration by a factor $f_i \in \mathbb{R}$ of a parameter $\theta_i \in \mathbb{R}$ defining one of the functions h or g of model 1, which modifies the dynamics of the system.*

Faults occur in system components, including sensors, actuators, or internal parts. A fault in a component $\mathcal{C}$, whose normal behavior is represented by the equation e, disrupts this equation. Consequently, this disruption may affect the diagnostic tests that include e in their equation support, or equivalently that include $\mathcal{C}$ in their component support.

▶ **Definition 7** (Fault signature matrix). *Given a model $\Sigma(z, x)$ and a set of diagnosis tests DT, a fault signature matrix (FSM) is a matrix where each column represents a component $\mathcal{C} \in COMP$ and each row a diagnosis test $\mathcal{T} \in DT$. A cell for a component $\mathcal{C} \in COMP$ and a diagnosis test $\mathcal{T} \in DT$ is set to 1 if and only if $\mathcal{C}$ belongs to the component support of $\mathcal{T}$, and 0 otherwise.*

The columns of the FSM provide the *fault signatures* of the faulty components while the lines provide the component supports of the diagnosis tests [3].

During the operation of the system, diagnosis tests are evaluated with the measurements. Their evaluation provides a value to the residuals and this value is binarized: 0 if the value is zero and 1 otherwise. If the residual of a test is 0, the diagnosis test passes while if it is different from zero, it fails. The evaluated residual binary vector is called the *observed signature*.

Diagnoses can be obtained in two ways [3]:

- By columns (the FDI way), i.e. by comparing the observation signature to the fault signatures,
- By rows (the DX way), i.e. by considering that at least one of the components in the component support of the failing tests must be incriminated (these provide *conflicts* in the sense of Reiter [11]).

To illustrate the above concepts and diagnosis process, consider the following two-tank system, illustrated in Fig. 1 [2, 10] whose objective is to provide water flow Q_o to a consumer area. The system comprises two tanks, $Tank_1$ and $Tank_2$, connected by a pipe. $Tank_1$ is filled by a pump $Pump$ to reach a nominal water level of $h_1 = 0.5m$. The water level in $Tank_1$ is regulated by a PI level controller $Cont$ that adjusts the inlet flow Q_p provided by the pump. The water flow Q_{12} between tanks is managed by valve V_b using an "ON/OFF" controller to maintain the water level h_2 within the range of $0.09m \leq h_2 \leq 0.11m$ in $Tank_2$. The outflow Q_o to a consumer area depends on the position of valve V_o, which is open in the nominal regime. The connecting pipe between the tanks is positioned at the bottom, equipped with valve V_b. The dynamic model of the non faulty behavior of the two-tank system is given by the two following equations:

$$Tank_1: \quad \dot{h}_1 = \frac{Q_p - C_{vb} \cdot \text{sgn}(h_1 - h_2)\sqrt{|h_1 - h_2|} \cdot U_b}{A_1} \tag{eq.1}$$

$$Tank_2: \quad \dot{h}_2 = \frac{C_{vb} \cdot \text{sgn}(h_1 - h_2)\sqrt{|h_1 - h_2|} \cdot U_b - C_{vo} \cdot \sqrt{h_2} \cdot U_o}{A_2} \tag{eq.2}$$

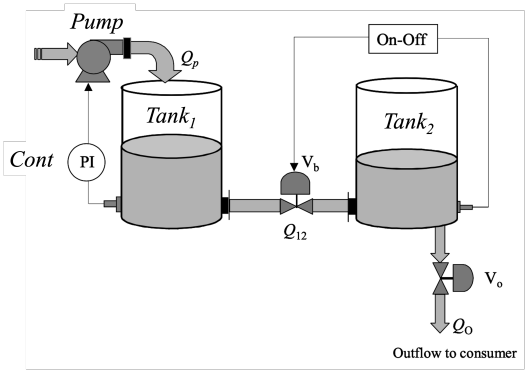


Figure 1 Two-tank system (from [2]).

We assume that Q_p is given by the output U_p of the PI controller, hence we have the following PI controller and pump equations:

$$Cont: \quad U_p = K_p(h_{1c} - h_1(t)) + K_i \int (h_{1c} - h_1(t))\, dt \tag{eq.3}$$

$$Pump: \quad Q_p = U_p. \tag{eq.4}$$

U_o and U_b represent the position of valve V_o and V_b respectively. These valves can only be open or closed, hence U_o and $U_b \in \{0, 1\}$. C_{vb} and C_{vo} are constant hydraulic flow coefficients, A_1 and A_2 are constant parameters of the tanks, h_{1c} is the target level for tank $Tank_1$, and K_p and K_i are constant parameters of the PI controller $Cont$.

Sensor equations are the following:

$$\text{Sensor } S_{h_1} \text{ of } h_1: \quad mh_1 = h_1 \tag{eq.5}$$

$$\text{Sensor } S_{h_2} \text{ of } h_2: \quad mh_2 = h_2 \tag{eq.6}$$

$$\text{Sensor } S_{Q_p} \text{ of } Q_p: \quad mQ_p = Q_p \tag{eq.7}$$

$$\text{Sensor } S_{U_b} \text{ of } U_b: \quad mU_b = U_b \tag{eq.8}$$

$$\text{Sensor } S_{U_p} \text{ of } U_p: \quad mU_p = U_p \tag{eq.9}$$

$$\text{Sensor } S_{U_o} \text{of } U_o: \quad mU_o = U_o. \tag{eq.10}$$

Given the available measurements, ARRs are easily obtained by substituting unknown variables in the behavior equations. We hence obtain the following four diagnosis tests:

$$\mathcal{T}_1: r_1 = -C_{vb}\,\mathrm{sgn}(mh_1 - mh_2)\sqrt{|mh_1 - mh_2|}\, mU_b + mQ_p - A_1\left(\frac{dmh_1}{dt}\right)$$

$$\mathcal{T}_2: r_2 = C_{vb}\,\mathrm{sgn}(mh_1 - mh_2)\sqrt{|mh_1 - mh_2|}\, mU_b - C_{vo}\cdot\sqrt{(mh_2)}\cdot mU_o - A_2\left(\frac{dmh_2}{dt}\right)$$

$$\mathcal{T}_3: r_3 = mU_p - K_p(h_{1c} - mh_1(t)) - K_i \int (h_{1c} - mh_1(t))\, dt$$

$$\mathcal{T}_4: r_4 = mQ_p - mU_p.$$

The components and the corresponding equations representing their behavior are given in Table 1. The equation and component supports of the four diagnosis tests $\mathcal{T}_1$ to $\mathcal{T}_4$ are given in Table 2.

Let us consider a set of possible faulty components, without need to precisely identify the faults nor their type:

Table 1 Components and their corresponding equations in the two-tank system.

$Tank_1$	$Tank_2$	$Cont$	$Pump$	S_{h_1}	S_{h_2}	S_{Q_p}	S_{U_b}	S_{U_p}	S_{U_o}
(eq.1)	(eq.2)	(eq.3)	(eq.4)	(eq.5)	(eq.6)	(eq.7)	(eq.8)	(eq.9)	(eq.10)

Table 2 Equation and component supports of the diagnosis tests.

Test	Equation support	Component support
$\mathcal{T}_1$	{(eq.1),(eq.5),(eq.6),(eq.7),(eq.8)}	$\{Tank_1, S_{h_1}, S_{h_2}, S_{Q_p}, S_{U_b}\}$
$\mathcal{T}_2$	{(eq.2),(eq.5),(eq.6),(eq.8),(eq.10)}	$\{Tank_2, S_{h_1}, S_{h_2}, S_{U_b}, S_{U_o}\}$
$\mathcal{T}_3$	{(eq.3), (eq.5), (eq.9)}	$\{Cont, S_{h_1}, S_{U_p}\}$
$\mathcal{T}_4$	{(eq.4),(eq.7),(eq.9)}	$\{Pump, S_{Q_p}, S_{U_p}\}$

- Faulty plant components, i.e., $Tank_1$ and $Tank_2$. These components being faulty would disrupt equation (eq.1) and equation (eq.2) respectively,
- Faulty controller $Cont$, which would disrupt equation (eq.3),
- Faulty actuator $Pump$, which would disrupt equation (eq.4),
- Faulty sensors S_{h_1}, S_{h_2}, S_{Q_p}, S_{U_b}, S_{U_p}, and S_{U_o} that would disrupt equations (eq.5) to (eq.10).

The FSM is given by Table 3.

Table 3 FSM of the two-tank system.

	$Tank_1$	$Tank_2$	$Cont$	$Pump$	S_{h_1}	S_{h_2}	S_{Q_p}	S_{U_b}	S_{U_p}	S_{U_o}
$\mathcal{T}_1$	1	0	0	0	1	1	1	1	0	0
$\mathcal{T}_2$	0	1	0	0	1	1	0	1	0	1
$\mathcal{T}_3$	0	0	1	0	1	0	0	0	1	0
$\mathcal{T}_4$	0	0	0	1	0	0	1	0	1	0

Assume that the actual mode of the two-tank system is $Cont$ faulty and all the other components normal. Also assume that the evaluation of the diagnosis tests $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, and $\mathcal{T}_4$ with measurements provides the observed signature is $(0, 0, 1, 0)^T$. The observed signature is matched against the faulty component signatures found int the FSM columns, which provides a unique one single diagnosis $\{Cont\}$, i.e. the PI controller.

3 Spectrum-based approach

Spectrum-based fault localization (SFL) is an approach based on probabilities considering different executions of a program. The underlying idea is to count how often statements are executed in passing and failing runs to compute a suspicious score for statements. For example, a statement that is only executed in passing runs can be considered least suspicious in causing a failure whereas a statement that is executed in all failing runs is most likely to be the reason behind wrong behavior. The first who carried out this idea were Jones and Harrold [5]. In their paper the authors introduced the basic concepts and their Tarantula tool. In the past decades, SFL has gained much attention in the debugging community as pointed out by Wong and colleagues [13].

In the following, we discuss the basic definitions and algorithms behind SFL considering an example from a previous publication of one of the authors [14]. Figure 2 shows a simple program comprising one class `Foo` and a method `foo`. The purpose of this program is to

```
1.  public class Foo {
2.   public static float a;
3.   public static float c;
4.   public static float pi = 3.14159;
5.   public static void foo (
6.              float d, int type) {
7.       if (type == 0) {      // Circle
8.          float r = d / 2;
9.          a = r * r * pi;
10.         c = d * pi;
11.      } else {
12.          if (type == 1) {  // Square
13.             a = d * d;
14.             c = 4 * d;
15.          } else {          // Error
16.             a = -1;
17.             c = -1;
18.          }
19.      }
20.   }
21. }
```

Figure 2 Simple program `Foo.foo` computing the circumference and the area of a circle and square depending on the second input (taken from [14]).

compute the circumference and the area of a circle and square. Table 4 shows typical tests we would apply for verifying the correctness of `Foo.foo`. In order to show how SFL computes the suspiciousness index, we need a fault version of a program, i.e., in our case `Foo.foo'`. In `Foo.foo'` we replace line 10 with `c = r * pi;` leading to a failing execution of test case T_1 from Table 4. Note that all other test cases are passing ones, i.e., delivering the expected values for both variables `a` and `c`.

To show SFL in action, we first define the notation of a *program spectrum* representing the execution of statements of a program for all test cases.

▶ **Definition 8** (Program spectrum). *Given a program* Π *and a test suite* TS*. A program spectrum is a matrix, where each line represents a statement of* Π *and each row a test case* $T \in TS$*. A cell for a statement* $S \in \Pi$ *and a test case* $T \in TS$ *is set to 1 if and only if the statement* S *is executed using test case* T*, and 0 otherwise.*

To access an element of the program spectrum in column i and row j, we write x_{ij}. The second important part of SFL is the error vector e. For each test case $T \in TS$ there is a corresponding entrance in e. This entrance is 1 if TS is a failing test case, i.e., the program is returning an unexpected output, and 0, otherwise.

In SFL the program spectrum and the error vector together are the *observation matrix*. In Table 5, we depict the observation matrix for `Foo.foo'`.

To compute the suspiciousness values for each statement, SFL, maps the observation matrix, i.e., the spectrum and the error vector, into 4 different kinds of metrics a_{ef} for each statement. Note that the a_{ef} represents the number of involvements of the given statement to executions or non-executions in passing and failing runs, where e is 1 if the statement is

Table 4 Test cases for program `Foo.foo` from Figure 2 (taken from [14].

Test	Fct. call	Expected result
T_1	`foo(1,0)`	`a=1.570795, c=3.14159`
T_2	`foo(1,1)`	`a=1.0, c=4.0`
T_3	`foo(1,2)`	`a=-1.0, c=-1.0`
T_4	`foo(0,0)`	`a=0.0, c=0.0`
T_5	`foo(0,1)`	`a=0.0, c=0.0`
T_6	`foo(0,2)`	`a=-1.0, c=-1.0`

Table 5 Applying SFL to `Foo.foo'` utilizing all 6 test cases.

	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
`7. if (type == 0) {`	1	1	1	1	1	1	0	0	5	1	0.408	2
`8. float r = d / 2;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`9. a = r * r * pi;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`10. c = r * pi;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`11. } else {`												
`12. if (type == 1) {`	0	1	1	0	1	1	1	1	4	0	0.000	3
`13. a = d * d;`	0	1	0	0	1	0	3	1	2	0	0.000	3
`14. c = 4 * d;`	0	1	0	0	1	0	3	1	2	0	0.000	3
`15. } else {`												
`16. a = -1;`	0	0	1	0	0	1	3	1	2	0	0.000	3
`17. c = -1;`	0	0	1	0	0	1	3	1	2	0	0.000	3
`18. }`												
`19. }`												
Error	1	0	0	0	0	0						

executed and 0 otherwise, and f is 1 if the test case is failing and 0 otherwise. Formally, we define a_{ef} for $e = 0, 1, f = 0, 1$ as follows:

$$a_{00}(j) = |\{i | x_{ij} = 0 \wedge e_i = 0\}|$$
$$a_{01}(j) = |\{i | x_{ij} = 0 \wedge e_i = 1\}|$$
$$a_{10}(j) = |\{i | x_{ij} = 1 \wedge e_i = 0\}|$$
$$a_{11}(j) = |\{i | x_{ij} = 1 \wedge e_i = 1\}|$$

For example, the $a_{10}(j)$ for a statement j indicates the number of passing test cases where j is executed, whereas $a_{01}(j)$ represents the number of failing test cases where j is not executed. SFL now takes these a_{ef} values to calculate a suspicious value for each statement. It is worth noting that there are a lot of different similarity coefficients described in literature. For example, Wong et al. [13] mentioned more than 31 of them. However, in the following we utilize the Ochiai coefficient, which is one that is often used in practice because of delivering good results.

The Ochiai coefficient for statement j is defined as follows:

$$c_O(j) = \frac{a_{11}(j)}{\sqrt{(a_{11}(j) + a_{01}(j)) \cdot (a_{11}(j) + a_{10}(j))}}$$

Note that the fraction's denominator in the Ochiai coefficient might become 0. In this case, c_O is defined to be 0. Note that the Ochiai coefficients for the `Foo.foo'` program and their ranks, defined from the coefficients ordering, are depecited in Table 5. We see that three statements have the same coefficient and, therefore, the same rank and cannot be distinguished by SFL. However, there are extensions of SFL also considering data dependencies, e.g., [14] leading to better results.

4 Fault Signature Matrix versus Spectrum-based Fault Localization

The FSM approach is used primarily in hardware systems to detect and isolate faults. It involves monitoring system inputs and outputs through diagnostic tests that are built from a model of the physical system considering the impact of faults to parts of the model, e.g., the residual equations.

The SFL approach is a method used in software engineering to identify the location of faults within software by analyzing the program's behavior during execution. In particular, it considers executed statements for computing a fault likelihood of statements.

This section aims to highlight the similarities between the FSM and SFL approaches. Although the first is dedicated to hardware systems and the second to software, the concepts and objects they manipulate are comparable and the underlying principles and techniques share common ground.

4.1 About tests

In software engineering and in SFL, a test case targets to execute a set of statements and test cases are applied to the real program, i.e. the program is run with specific inputs for which the program output is known.

On the other hand, when diagnosing a physical system that is in operation, it is generally not possible to apply specific inputs to the system except in particular cases in which active diagnosis [9] is viable but we do not consider this case in this paper. So, the inputs and outputs of the physical system are taken as they are applied to the system by the controller, designed to achieve the current control objectives. In the FSM approach, diagnosis tests – or residual generators – are obtained from a mathematical model that is assumed to be a proper representation of the physical system. They establish a relation between observable variables. The physical quantities corresponding to observable variables that are linked by diagnosis tests are connected through a subset of components of the system that establish a path between these variables. These paths are identified through the equations representing these components. Indeed, a diagnostic test results from a process of eliminating non-observable variables, which can be implemented in the simplest case through sequential substitution [12]. When a fault occurs in a physical component $\mathcal{C}$ whose normal behavior is represented by the equation e, the inputs to the physical system that activate $\mathcal{C}$ generate outputs that should disrupt equation e and hence all the tests that have e in their equation support, or equivalently $\mathcal{C}$ in their component support. These test would then generate a non zero residual.

Let us remember that a test case in the SFL framework is noted as T and a test in the FSM framework as $\mathcal{T}$. Although the differences above in their design, they share the following commonalities:

- they are supported by a subset of statements (SFL) and a subset of components (FDI), respectively,
- the tests pass when the statements or the components represented by the equations that support the tests are normal,
- the tests may fail when the statements or the components represented by the equations that support the tests are faulty.

4.2 FDI fault signature matrix vs SFL observation matrix

Interestingly, it can be noticed that transposing the FSM of Table 3, and adding a row for the residual binary vector resulting from the evaluation of the tests, i.e. for the *observed signature* (called the *error vector* in SFL), we obtain an object that has the same semantics as the SFL observation matrix (recalled in Table 7) as displayed below in Table 6.

Table 6 Transp. FS matrix.

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$
$Tank_1$	1	0	0	0
$Tank_2$	0	1	0	0
$Cont$	0	0	1	0
$Pump$	0	0	0	1
S_{h_1}	1	1	1	0
S_{h_2}	1	1	0	0
S_{Q_p}	1	0	0	1
S_{U_b}	1	1	0	0
S_{U_p}	0	0	1	1
S_{U_o}	0	1	0	0
obs-error	?	?	?	?

Table 7 SFL observation matrix.

	T_1	T_2	T_3	T_4	T_5	T_6
`7. if (type == 0) {`	1	1	1	1	1	1
`8. float r = d / 2;`	1	0	0	1	0	0
`9. a = r * r * pi;`	1	0	0	1	0	0
`10. c = r * pi;`	1	0	0	1	0	0
`12. if (type == 1) {`	0	1	1	0	1	1
`13. a = d * d;`	0	1	0	0	1	0
`14. c = 4 * d;`	0	1	0	0	1	0
`16. a = -1;`	0	0	1	0	0	1
`17. c = -1;`	0	0	1	0	0	1
obs-error	?	?	?	?	?	?

At this point, we can already say that the processes used by FSM and by SFL could be applied interchangeably on the FSM matrix or the observation matrix to obtain the diagnoses. In particular, we would be able to use SFL to come up with the evaluation of the suspiciousness value for each component and a ranking, which might be an improvement in some practical cases. The results of applying the SFL Ochiai method to the two-tank system for an observed signature/error vector $(1, 0, 0, 0)^T$ – that we denote *obs-error* in the following – are given in Table 8.

Whenever an error vector is equivalently represented in a row, the suspiciousness index becomes 1.0. Hence, such a row would be ranked on the top indicating that the corresponding component is faulty. For example, in Table 8 the error vector is equivalent to the vector stored in the row of component $Tank_1$, which is ranked first in this example. This result is the same as the one we would obtain using the FSM method. However, there is a difference. If this error vector is not visible to 100%, the ranking of faults might help in case of double and triple faults. In such a case the ordinary FSM would not deliver any output and would have to be extended with the signatures of multiple faults to be effective [3]. Hence, the use of SFL could provide another way to handle multiple faults, avoiding the burdensome task of enumerating their signatures. This shows that utilizing the ideas from SFL could improve diagnosis reasoning of hardware. Conversely, directly comparing the error vector with the *execution signature* of each statement – defined as the program spectrum rows by analogy to fault signatures in FSM – using a specific metric such as Hamming distance (commonly applied in FSM) could also be a viable approach. Moreover, the methodology used in the FSM framework to construct diagnostic tests by eliminating unmeasured variables could offer useful insights for designing tests in software diagnostics.

4.3 Summary

The FSM and the SFL approaches share many commonalities considering components as program statements and the residual generators (i.e., diagnosis tests) as test cases. Considering an FSM as a program spectrum allows for computing a suspiciousness values (e.g., the Ochiai coefficients) and a fault ranking. The advantage of this mapping is that

Table 8 Applying SFL to the Two-tanks system.

	T_1	T_2	T_3	T_4	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	3	0	0	1	1.000	1
$Tank_2$	0	1	0	0	2	1	1	0	0.000	4
$Cont$	0	0	1	0	2	1	1	0	0.000	4
$Pump$	0	0	0	1	2	1	1	0	0.000	4
S_{h_1}	1	1	1	0	1	0	2	1	0.5774	3
S_{h_2}	1	1	0	0	2	0	1	1	0.7071	2
S_{Q_P}	1	0	0	1	2	0	1	1	0.7071	2
S_{U_b}	1	1	0	0	2	0	1	1	0.7071	2
S_{U_p}	0	0	1	1	1	1	2	0	0.000	4
S_{U_o}	0	1	0	0	2	1	1	0	0.000	4
obs-error	1	0	0	0						

also double and triple faults of hardware can be handled directly. On the other hand, the concept of an "observed signature" in FSM could be translated into an "execution signature" in SFL, and the methodology for constructing diagnostic tests in FDI could provide valuable guidance for test design in software diagnostics.

Based on these suggestions, we further discuss an integrated approach that allows a combined diagnosis of hardware and software in the following section.

5 Integrating FSM and SFL

In this section, we discuss how FSM and SFL can be combined together. For this purpose, we use a small example to illustrate the basic underlying ideas behind the method integration.

Let us consider the two tanks system and assume that the control U_p provided by the controller to control the pump is now provided by the program `Foo.foo2` that is a slightly modified version of the program `Foo.foo`. The PI controller is replaced by a controller delivering a control U_p proportional to the weight of water in the tank, which is equal to the area of the tank times the height of water. The tank may be circular or rectangular. Figure 3 depicts the program `Foo.foo2`.

In the model of this cyber-physical system, equation (eq.3) modeling the PI controller is now replaced by the code of the program `Foo.foo2`. We then can integrate the FSM and SFL observation matrices as shown in Table 9, where the grey row for *Cont* indicates that this row must be removed.

Note that test $\mathcal{T}_3$ that was previously supported by the component *Cont* (ghost line in gray) is now supported by the statements that are executed to deliver U_p (marked in bold in Table 9).

Let us consider two scenarios for this cyber-physical system:

- *Scenario 1*: the two-tank cyber-physical system suffers a single fault in the software `Foo.foo2` controller in statement 9, i.e.,`a=r*pi` instead of `a=r*r*pi`, which is marked in bold red in Table 9.
- *Scenario 2*: the two-tank cyber-physical system suffers a double fault, including the above fault in statement 9 plus a fault in the hardware sensor S_{h_1}.

```
1.  public class Foo {
2.   public static float a;
3.   public static float Up;
4.   public static float pi = 3.14159;
5.   public static void foo2 (
6.              float d, int type, float h) {
7.       if (type == 0) {      // Circle
8.          float r = d / 2;
9.          a = r * r * pi;
10.         Up = a * h;
11.      } else {
12.          if (type == 1) {  // Square
13.             a = d * d;
14.          Up = a * h;
15.          } else {          // Error
16.             a = -1;
17.             Up = -1;
18.          }
19.      }
20.   }
21. }
```

Figure 3 Program `Foo.foo2` computing the control of the pump U_p in the two-tank system for a circle or square tank shape depending on the second input.

5.1 Scenario 1: single fault

Scenario 1: the two-tank cyber-physical system suffers a single fault in the software `Foo.foo2` controller in statement 9, i.e.,`a=r*pi` instead of `a=r*r*pi`, which is marked in bold red in Table 9.

The application of the two methods, FSM and SFL to the two-tank system in Scenario 1 is illustrated in Table 10. The obs-error vector, resulting from the evaluation of $\mathcal{T}_1$ to $\mathcal{T}_4$ and from running test cases T_1 to T_6 is given by $(0, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$.

According to the FSM method, the obs-error vector matches the fault signatures of the single faults {stat. 8}, {stat. 9} and {stat. 10} ("stat." is used as an abbreviation of "statement"), which definitively indicates a fault in the software controller but does not discriminate between the three statements. The corresponding rows are colored green in Table 10.

According to the SFL method, the Ochiai coefficients provide a ranking that also puts at the first rank all three single faults {stat. 8}, {stat. 9} and {stat. 10}.

The results of the two methods are hence consistent. Note that these results are quite understandable because the actual fault, `a=r*pi` instead of `a=r*r*pi`, is not discriminable from a fault in statement 8, e.g. a wrong value for `d` in `r=d/2` or a fault in statement 10, e.g., a wrong value for `h` in `Up=a*h`.

5.2 Scenario 2: double fault

Scenario 2: the two-tank cyber-physical system suffers a double fault, including the above fault in statement 9 plus a fault in the hardware sensor S_{h_1}.

Table 9 FSM and SFL observation matrices integrated.

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6
$Tank_1$	1	0	0	0	0	0	0	0	0	0
$Tank_2$	0	1	0	0	0	0	0	0	0	0
$Cont$	0	0	1	0	0	0	0	0	0	0
$Pump$	0	0	0	1	0	0	0	0	0	0
S_{h_1}	1	1	1	0	0	0	0	0	0	0
S_{h_2}	1	1	0	0	0	0	0	0	0	0
S_{Q_p}	1	0	0	1	0	0	0	0	0	0
S_{U_b}	1	1	0	0	0	0	0	0	0	0
S_{U_p}	0	0	1	1	0	0	0	0	0	0
S_{U_o}	0	1	0	0	0	0	0	0	0	0
`7. if (type == 0) {`	0	0	**1**	0	1	1	1	1	1	1
`8. float r = d / 2;`	0	0	**1**	0	1	0	0	1	0	0
`9. a = r * pi;`	0	0	**1**	0	1	0	0	1	0	0
`10. Up = a * h;`	0	0	**1**	0	1	0	0	1	0	0
`12. if (type == 1) {`	0	0	0	0	0	1	1	0	1	1
`13. a = d * d;`	0	0	0	0	0	1	0	0	1	0
`14. Up = a * h;`	0	0	0	0	0	1	0	0	1	0
`16. a = -1;`	0	0	0	0	0	0	1	0	0	1
`17. Up = -1;`	0	0	0	0	0	0	1	0	0	1
obs-error	0	0	**1**	0	**1**	0	0	**1**	0	0

Dealing with multiple faults with the FSM method would require to generate their fault signature, which is commonly performed by applying a logical OR to the single fault signatures. In our scenario, according to the FSM method, the obs-error vector would then match the fault signatures of the double faults {stat. 8, S_{h_1}}, {stat. 9, S_{h_1}}, and {stat. 10, S_{h_1}}, identifying correctly and unambiguously the sensor S_{h_1} and the controller while obviously leaving undiscriminated {stat. 8}, {stat. 9} and {stat. 10}. The corresponding rows are colored green in Table 11.

According to the SFL method, the Ochiai coefficients provide a ranking that also puts at the first rank all three the three double faults {stat. 8, S_{h_1}}, {stat. 9, S_{h_1}}, and {stat. 10, S_{h_1}}.

The results of the two methods are here again hence consistent.

5.3 Summary

The two scenarios solved using the FSM and SFL methods within the integrated framework yield consistent results, demonstrating the compatibility of the two approaches and highlighting the effectiveness of the integrated framework in diagnosing cyber-physical systems.

6 Conclusion

In this paper, we have undertaken a comparative analysis of two key Fault Detection and Isolation (FDI) methodologies: the Faults Signature Matrix approach (FSM), commonly applied in hardware diagnostics, and the Spectrum-based fault localization (SFL) method, prevalent in software systems. Despite their distinct application domains, our analysis has shown that both approaches share fundamental principles. By leveraging the strengths of each method – structured pattern recognition in hardware systems and statistical analysis in software fault localization – there is significant potential to develop hybrid FDI strategies that address the challenges of cyber-physical systems, which increasingly integrate both hardware and software components.

Table 10 Single fault scenario – FSM and SFL observation matrices integrated (left of the table), Ochiai results (right of the table).

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
$Tank_2$	0	1	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
$Pump$	0	0	0	1	0	0	0	0	0	0	6	3	1	0	0,0000	5
S_{h_1}	1	1	1	0	0	0	0	0	0	0	5	2	2	1	0,3333	4
S_{h_2}	1	1	0	0	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{Q_p}	1	0	0	1	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{U_b}	1	1	0	0	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{U_p}	0	0	1	1	0	0	0	0	0	0	6	2	1	1	0,4082	3
S_{U_o}	0	1	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
stat. 7	0	0	1	0	1	1	1	1	1	1	3	0	4	3	0,6547	2
stat. 8	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 9	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 10	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 12	0	0	0	0	0	1	1	0	1	1	3	3	4	0	0,0000	5
stat. 13	0	0	0	0	0	1	0	0	1	0	5	3	2	0	0,0000	5
stat. 14	0	0	0	0	0	1	0	0	1	0	5	3	2	0	0,0000	5
stat. 16	0	0	0	0	0	0	1	0	0	1	5	3	2	0	0,0000	5
stat. 17	0	0	0	0	0	0	1	0	0	1	5	3	2	0	0,0000	5
obs-error	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**						

Table 11 Double fault scenario – FSM and SFL observation matrices integrated (left of the table), Ochiai results (right of the table).

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
$Tank_2$	0	1	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
$Pump$	0	0	0	1	0	0	0	0	0	0	4	5	1	0	0,0000	6
S_{h_1}	1	1	1	0	0	0	0	0	0	0	5	2	0	3	0,7746	1
S_{h_2}	1	1	0	0	0	0	0	0	0	0	5	3	0	2	0,6325	2
S_{Q_p}	1	0	0	1	0	0	0	0	0	0	4	4	1	1	0,3162	5
S_{U_b}	1	1	0	0	0	0	0	0	0	0	5	3	0	2	0,6325	2
S_{U_p}	0	0	1	1	0	0	0	0	0	0	4	4	1	1	0,3162	5
S_{U_o}	0	1	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
stat. 7	0	0	1	0	1	1	1	1	1	1	1	2	4	3	0,5071	3
stat. 8	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 9	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 10	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 12	0	0	0	0	0	1	1	0	1	1	1	5	4	0	0,0000	6
stat. 13	0	0	0	0	0	1	0	0	1	0	3	5	2	0	0,0000	6
stat. 14	0	0	0	0	0	1	0	0	1	0	3	5	2	0	0,0000	6
stat. 16	0	0	0	0	0	0	1	0	0	1	3	5	2	0	0,0000	6
stat. 17	0	0	0	0	0	0	1	0	0	1	3	5	2	0	0,0000	6
obs-error	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**						

The findings of this investigation highlight the complementary nature of these two methodologies, suggesting that combining their capabilities could lead to more robust and comprehensive diagnostic systems. This synergy could result in an integrated diagnostic framework capable of managing the intricacies of both hardware and software failures in modern, complex systems.

The theoretical insights gained from this study lay the groundwork for future research into hybrid FDI systems, with practical applications that span a wide range of technological platforms. By advancing the integration of these diagnostic techniques, we aim to contribute to the development of more reliable, efficient, and adaptable fault detection systems, ultimately improving the resilience and performance of cyber-physical systems.

7 References

References

1 Michele Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. prentice Hall Englewood Cliffs, 1993.

2 B Ould Bouamama, R Mrani Alaoui, P Taillibert, and M Staroswiecki. Diagnosis of a two-tank system. *Intern Report of CHEM-project, USTL, Lille, France*, 2001.

3 M-O Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004. doi:10.1109/TSMCB.2004.835010.

4 Carine Jauberthie, Nathalie Verdière, and Louise Travé-Massuyès. Fault detection and identification relying on set-membership identifiability. *Annual Reviews in Control*, 37(1):129–136, 2013. doi:10.1016/J.ARCONTROL.2013.04.002.

5 J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings ASE'05*, pages 273–282. ACM Press, 2005.

6 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):197–206, 2007. doi:10.1109/TSMCA.2007.909555.

7 Ronald John Patton and Jie Chen. Design methods for robust fault diagnosis. *Control Syst. Robot. Autom*, 16:84–111, 2009.

8 Vicenç Puig and Masoud Pourasghar. Fault diagnosis using set-membership approaches. *Fault Diagnosis of Dynamic Systems: Quantitative and Qualitative Approaches*, pages 237–261, 2019.

9 Ivo Punčochář and Jan Škach. A survey of active fault diagnosis methods. *IFAC-PapersOnLine*, 51(24):1091–1098, 2018.

10 Joseba Quevedo, Helem Sánchez, Damiano Rotondo, Teresa Escobet, and Vicenç Puig. A two-tank benchmark for detection and isolation of cyber attacks. *IFAC-PapersOnLine*, 51(24):770–775, 2018.

11 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

12 Christofer Sundström, Erik Frisk, and Lars Nielsen. Selecting and utilizing sequential residual generators in fdi applied to hybrid vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(2):172–185, 2013. doi:10.1109/TSMC.2013.2248147.

13 W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016. doi:10.1109/TSE.2016.2521368.

14 Franz Wotawa. Surveying and generalizing methods for combining dynamic slicing with spectrum-based fault localization. In *In* Proceedings of the International Workshop on Principles of Diagnosis (DX), Loma Mar, CA, USA, September 2023.

Challenges for Model-Based Diagnosis

Ingo Pill
Institute of Software Technology, Graz University of Technology, Austria

Johan de Kleer
c-infinity, Mountain View, CA, USA

Abstract

Since the seminal works by Reiter and de Kleer and Williams published in the late 80's, Model-based Diagnosis has been a significant area of research. This has been motivated by the fact that MBD assists us in tackling a challenge that we face almost on a daily basis, i.e., by MBD allowing us to reason in a structured manner about the root causes for some encountered problem. MBD achieves this in an intuitive, complete and sound way, based on the central idea of investigating the compliance of some observed behavior with a model that describes how a system should behave – given this or that input scenario and parameter set. Over the last 40 years, MBD has been adopted for a multitude of applications, and we saw the emergence of a diverse set of algorithmic, optimizations, as well as extensions to the initial theoretical concepts.We argue that MBD remains highly relevant, with numerous scientific challenges to tackle as we face increasingly complex diagnostic problems. We discuss several such challenges and suggest related topics for PhD theses that have the potential to significantly contribute to the state-of-the-art in MBD research.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Model-based Diagnosis, Diagnosis, Algorithms

Digital Object Identifier 10.4230/OASIcs.DX.2024.6

Acknowledgements We would like to thank the DX community for discussions on topics related to this paper. Among other people, this specifically includes (in a.o.) *Gautam Biswas, Alessandro Cimatti, Alex Feldman, Kai Goebel, Meir Kalech, Ion Matei, Oliver Niggemann, Markus Stumptner, Louise Trave-Massuyes, Franz Wotawa, and Marina Zanella.* This article was influenced also by discussions led at the Dagstuhl Seminar 24031 *Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis*, where we would like to thank the organizers as well as *Schloss Dagstuhl Leinbniz-Zentrum für Informatik* for organizing and hosting this seminar.

1 Motivation

Model-based Diagnosis (MBD) [51, 11, 12] has been a significant area of research for almost 50 years. It is based on the central idea that diagnosis of a system can be achieved by analyzing how deviations in behavior can be explained by undesired changes in the model of the system. A key advantage of model-based diagnosis is that it relies solely on the formal description of the design of the system, not diagnostic trees, expert systems or human experience. Implementing a concept of reasoning from first principles, we construct the most coherent explanation between our observations on what happened, and our knowledge about how a system should behave. The implicit assumption is that the other parts of a system, i.e., those whose behavior is inconsistent with the information, are those where something went wrong. Being completely flexible in our choice, parts can be system components, measurements, or any other knowledge we have available for exploitation in our diagnostic reasoning process.

MBD is very flexible and thus allows us to focus the diagnostic analysis tailored to the specific context at hand. For a failed regression test, for example, we might want to restrict our focus to those system parts that were most recently changed as well as on components


© Ingo Pill and Johan de Kleer;
licensed under Creative Commons License CC-BY 4.0



interacting with them – a technique called program slicing [6] in the context of software engineering. Or we might want to focus only on specific temporal phases in some temporal sequence, like a system's initialization.
In contrast to expert systems, one of MBD's distinct advantages is that if we change or revise individual system parts, we only need to remodel those parts – rather than having to redevelop our entire knowledge base concerning the effects of individual faults and their combinations on the behavior of the entire system.

MBD has proven to be a powerful technique and has been adopted for a variety of problem settings and domains. Nevertheless, there remain open challenges that concern fundamental topics as well as the exploitation of connections to complementing research fields. A prominent example for the latter would be achieving resilience in an autonomous system, which requires diagnostic information about a problem such as to successfully mitigate its effects. In this paper, we articulate a selection of these challenges and encourage researchers to address some of them in their future research.

We will have undoubtedly missed certain challenges to MBD, and we would also like to point out that fellow researchers have similar ideas. So it is not the uniqueness of the challenges that represents the contribution of this paper. Our intention is rather to draw the scientific community's attention to these challenges in order to revive old discussions and trigger new ones on how we can address them in order to propel MBD research to be able to have far wider applicability for the problems of today and tomorrow. We intend to maintain this paper as a living document and will add and modify sections as we learn. So we request the interested reader to send us their comments and inputs on these challenges and we will update this paper accordingly.

2 Preliminaries: a Crash Course in Model-based Diagnosis

While we cover only the very basics, we still aim to provide the interested reader with the relevant background information for our discussions – offering also citations of relevant and more detailed publications.

MBD [51, 12] is a fundamental diagnosis framework that compares information about what happened (in the form of a set of observations *OBS*) with a description of the system's supposed behavior (in the form of a system description *SD*). There is no specific required format for expressing *OBS* or *SD*, but it includes some sort of reasoning engine that allows us to verify whether *OBS* lives up to the expectations covered in *SD*– similar to the task and purpose of a test oracle [46]. Formally, *OBS* should be consistent with *SD*, so that $OBS \wedge SD$ is satisfiable.

The innovation of MBD is, not the detection of an inconsistency but that it allows us to *reason about what went wrong* when *OBS* is inconsistent with *SD*. That is, implementing a concept of reasoning from first principles, MBD constructs maximum sets of information from *SD* that are consistent with *OBS*– implicitly assuming that *OBS* is correct. Traditionally, MBD algorithms [51, 12, 18, 41, 61] have been designed to be not only sound but also complete, in that they can derive all such maximum sets. The complements (in *SD*) of these maximum sets define diagnoses. That is, those parts that if they were to deviate from their nominal behavior would explain the observed scenario. There are also incomplete algorithms, i.e., algorithms that might shed some solutions in their search, while guaranteeing, for instance, that they would still find a sound diagnosis of minimal cardinality [14].

Reiter's theory of diagnosis [51] defines the model-based diagnosis of a system as follows (cf. also [12]): A system description *SD* captures the behavior of a set of interacting components *COMP* such that *SD* contains sentences $\neg AB(c_i) \Rightarrow NominalBehavior(c_i)$ en-

coding a component's behavior under the assumption that it is not operating abnormally. That is, the assumption predicate $AB(c_i)$ triggers a component c_i's abnormal behavior, and *NominalBehavior* defines its correct behavior in first order logic. Since there are no definitions regarding abnormal behavior, the basic approach is considered to use a *weak fault model* (WFM). Given some actually observed system behavior *OBS*, a system is recognized to be faulty, if and only if $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP\}$ is inconsistent. A minimal diagnosis for a diagnosis problem (*SD*, *COMP*, *OBS*) can then be defined as follows:

▶ **Definition 1.** *A* minimal diagnosis *for (SD, COMP, OBS) is a subset-minimal set* $\Delta \subseteq COMP$ *such that* $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ *is consistent.*

Reiter as well as de Kleer and Williams proposed to compute (all) the minimal diagnoses as the minimal hitting sets of the set of (not necessarily minimal) conflicts for (*SD*, *COMP*, *OBS*). This implements the idea that we indeed have to resolve such conflicts that individually express the fact that it is not possible that a specific set of components all work correctly when considering *OBS*. Thus, a conflict basically describes a set of assumptions that components work correctly that is in conflict with the observations *OBS*:

▶ **Definition 2.** *A* conflict *CS for (SD, COMP, OBS) is a set* $CS \subseteq COMP$ *such that* $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in CS\}$ *is inconsistent. If no proper subset of CS is a conflict set, CS is a minimal conflict.*

Using a theorem prover or solver capable of returning conflicts, we can compute the conflicts and diagnoses on-the-fly – using a variety of algorithms. Extensions of the first algorithms by de Kleer and Williams [12], Reiter [51], or Greiner et al. [18], are able to avoid redundancies in the computation [41], optimize the memory bloat associated with the open node list [54], and considered the problem from a different point-of-view such as to improve the computation [22]. In addition to new conflict-driven algorithms, researchers have also published *brute force* approaches that compute diagnoses directly in a solver, without explicitly computing conflicts [27, 36]. While such approaches were assumed in Reiter's seminal work [51] to show bad performance, comparisons showed that with today's computation hardware and recent developments in the context of SAT and constraint solvers, they are indeed viable. In particular, it was observed that they seem to offer performance that is at least on par with the original conflict driven approaches [35].

When deploying a *strong fault model* (SFM) MBD approach, we need to define the abnormal behavior – like a stuck-at-1 fault for a logic gate [13]. Depending on the implementation, the abnormal predicates from above would then be replaced by one or more variables that select in which mode a component is supposedly in. Although these additional modeling efforts result in a larger model, they offer the advantage of more specific diagnoses, identifying the actual fault modes of components. Such information is precious insofar as it eases the debugging process significantly by describing the scenario in more details. Furthermore, with specific models – like when diagnosing specifications – a corresponding diagnosis even reports a repair [40] like that we should have used operator F instead of G in a certain place of a formula. As a non-negligible downside, however, the search space for diagnoses grows significantly. That is, with n the number of components/selector variables and m the maximum number of modes for any assumption, it grows from 2^n to $\mathcal{O}(m^n)$.

In our descriptions, we decomposed *SD* into components, a method commonly used in Model-Based Design (MBD) literature. No such restriction in required. We can use any subset of *SD* as a block and associate a health state or abnormal predicate with it. Furthermore, MBD is also not limited to using any non-temporal [51, 12] or temporal [40]

logic for capturing *SD*. Indeed also other formats, including Petri Nets [9], various kinds of automata [32] or their symbolic implementation can be used (the latter two being close to temporal logics). In particular, we only require the means for checking the consistency of *OBS* with *SD* under the assumptions defined by a diagnosis (and ideally for computing conflicts for the negative case) for being able to employ MBD (see also [51]). We can thus also easily employ SFM MBD using a continuous Modelica model and an oracle that can consider aspects like noisy sensors when checking whether *OBS* matches *SD* for a diagnosis Δ.

Regarding the use of automata for *SD* in a run-time verification context, we would also like to direct interested readers to research on diagnosing discrete event systems There, we often capture faults via unobservable faulty transitions and then investigate means for detecting their presence via inspecting a DES's language and the corresponding observable traces [57]. Such execution-related aspects have been considered also for generating diagnostic hypotheses for hybrid systems [34]. Further, more detailed discussions of MBD, like that considering new observations is non-monotonic, are available in the seminal works [51, 12].

3 Critical Directions for Future DX Research

While MBD has been an active research area since the late 80s, the motivation of being able to formally reason about the root causes for some encountered issue never lost in attractiveness or relevance. The exponential search space and potentially large models entail a certain computational complexity though, so that there is a continuous need to improve on the efficiency of available algorithms, and look for methods to speed up the computations, e.g., by exploiting domain knowledge [56, 42]. Furthermore, there is a constant demand for new MBD engines that would enable us to accommodate new types of models.

In the remainder of our paper, we list of further challenges for MBD. These challenges complement the natural ones as introduced above by focusing on specific questions about certain important aspects of MBD. Before motivating and discussing these challenges in individual subsections, let us compile a brief, comprehensive list first:

- Failure of function, not of a component is the core problem
- The continuous and discrete worlds were never adequately integrated
- Fault propagation
- Wear/component degradation
- Every model is an approximation
- Getting our hands around resilience and how to measure it
- Attacks or failure?
- Improving physical reasoning in MBD
- Exploiting synergies in terms of hypothesis discrimination and transferable insights when having access to multiple system instances
- Parallel computations with multiple cores and hardware acceleration
- Improving diagnostic information provided to human operators and autonomous control
- MBD as a design tool, or "Redesign by MBD"
- Exploiting the full power of integrating symbolic with sub-symbolic computations
- Where does the diagnosis model come from?

While this list is far from being complete, it targets algorithmic, systematic, usability-centered, and other aspects of MBD – which highlights the complexity and diversity of research on MBD. The topics illustrate also how tightly MBD connects and integrates with topics from V&V, debugging, control and system design. Please note that although there are many more topics of interest, we believe that addressing the presented challenges will significantly broaden MBD's applicability to current and future problems.

Each of these challenges merits in-depth discussion in its own dedicated paper, but our intention is to provide an overview. In the following subsections, we thus offer a brief description and motivation for each challenge. We furthermore point out topics for PhD theses that we could imagine to contribute significantly to solving the challenges.

3.1 Failure of Function

When your TV goes black, a repair person will not immediately go to the schematic and analyze what hard- or software component might have changed its behavior (e.g., becoming open or shorted for some electronic part). They will not start by looking at the sound circuit, because they know it cannot cause the black screen. The repair person has and draws on a common-sense intuitive understanding of TVs such that they know the sound circuit cannot cause a problem in the picture circuits. If they find a faulty transistor in the remote control, they will not get distracted by it because even though it is a deviation from the design, it cannot cause the observed symptoms. This topic has received very little attention, yet is fundamental to how humans troubleshoot. This is a very hard research topic because it requires developing a representation of human diagnostic common sense.

From a technical perspective, as humans we intuitively create sort of a dependency graph that captures also failure modes and their effects (and we do this at various abstraction levels), so that we can make quick decisions on a very abstract level without having to consider the entire system in all its detail. Creating technical models that can achieve this, while maintaining completeness of the reasoning process and being fully aware of the potentials and limitations of a specific model, is a severe challenge. Some initial steps have been made, e.g., in the direction of deriving dependency graphs and dependent failure descriptions [59]. The rules/abstract models learned in [29] could be interpreted as very abstract models that describe a systems abstract functionality. A full solution for this challenge could, however, substantially propel research on safety and resilience aspects of autonomous systems, as well as provide important stimuli for research on the efficient and effective control of complex system-of-system architectures.

- Identifying and Learning Abstract Concepts, Dependencies, and Interconnections in a System's Behavior for Their Exploitation in Model-Based Diagnosis
- An Approach to Hierarchical MBD Enabled by Automated Model Abstraction and Refinement Based on Capturing Functional Dependencies
- A Distributed Approach to MBD of Systems-of-Systems Exploiting the Exchange of Abstract System Models For Capturing Functional Interconnections
- Hierarchical and Dynamic MBD for Resilient Resource-Constrained Embedded Systems
- Developing a formal model of human common-sense diagnostic reasoning

3.2 Continuous and Discrete

A typical system has both continuous (e.g., resistor) and discrete (e.g., relay) components. Model-based diagnosis has made significant progress for systems that are purely continuous or purely discrete. Discrete components such as logic gates can be analyzed through causal signal propagation. Continuous systems on the other hand typically do not have direct causal paths. To analyze such systems requires solving simultaneous equations. It is much more difficult to determine causality in such systems because in the worst case every component can cause the observed symptoms. Fortunately, various techniques have been developed to extract some of the causality from continuous systems through techniques such as parameter estimation and structural analysis from the FDI community. However, systems that include both discrete and continuous models remain very challenging.

Such combined models might also surface in scenarios where we use different abstraction levels for describing the individual system parts, e.g., when aiming to optimize our computations. That is, some electronic part might be represented at a logic level, while we need to consider other components at the voltage or energy level for the problem under investigation, even if both are logic gates.

- An Approach to Modeling Systems for MBD that Unites Discrete and Continuous Model Parts and Supports Scaling the Abstraction Level of Individual Parts
- Capturing and Extracting Causality in Analog and Mixed-Signal Designs for Efficient Mode-Based Diagnosis
- Optimizing Model-Based Diagnosis of Mixed-Signal Designs

3.3 Fault Propagation

Fault propagation is rarely modeled in MBD approaches. The output stage of an amplifier may develop an internal short but still function ok. This now overloads the prior amplifier stage. The prior stage now blows its output transistor, causing the overall system to completely fail. MBD will notice the pre-amplifier has failed and repairs it. But it won't take long for it to blow again. An experienced diagnostician would ask themselves the question "Why did the output transistor blow", and understand that the root problem is prior stage not the output stage. They have a model that if a transistor is driving more current than designed, it will fail soon. A simpler version of this issue arises when your home blows a fuse. Sure, replacing a fuse will most probably restore your lights momentarily. But the ultimate cause was a dangling bare wire in your attic.

Previous work like the dependency graphs and dependent failure descriptions computed in [59] made first steps in this direction. With the rising demand for autonomous dependable systems, we however can easily see that we need more elaborate solutions that take details like effects from temporal fault sequences into account, support a continuous refinement during system operation – for example when recognizing that the current model is insufficient – and support resilient autonomous control in hypothesis discrimination [4, 20], fault mitigation [25, 67, 24] and contingency planning [66, 52].

- A Model for Capturing Fault Propagation for MBD Purposes and Deriving Appropriate Inspection and Debugging Strategies
- Enhanced Automated Fault Mitigation in Autonomous Systems by Augmenting MBD With Continuously Refined and Updated Fault Propagation Models
- Integrating Fault Propagation and Prognosis for Exploitation in MBD for Optimizing Resilient Autonomous Control

3.4 Component Degradation

MBD as it was defined originally [51, 12], employs a simple Boolean notion of whether a system's behavior captured in *OBS* adheres to the nominal one described in *SD*. In particular, a component is working either correctly or faulty. When incorporating fault modes in our reasoning [13], we're looking at faulty behavior in more detail. That is, in an SFM model we describe how a component can fail and a diagnosis then tells us in which fault mode a component is supposedly in. When incorporating individual fault models, we can thus reason about how a component failed – not how badly, but how. We are, however, not aware of MBD research that considers the positive case in more detail. That is, approaches that assess how good the current status of a system (and its components) is, and its remaining useful life. Currently, and taking aspects like component degradation into account, MBD

thus utterly fails to assist us in answering the question: is there some specific component that is likely to fail shortly? The PHM community has various models of remaining useful life (such as utilized in [25]), but none are compatible with MBD. [28] incorporates degradation models in Modelica, but the actual physics based calculation of wear have to be performed by an ad-hoc supplied physics model.

In the context of runtime-verification, a.k.a. monitoring, such considerations have been made. In particular, monitors generated for the Signal Temporal Logic (STL) [16] – which is a logic that merges continuous time and continuous signal values with the basic concepts of Amir Pnueli's Linear Temporal Logic (LTL) [49] – can consider specific quantitative semantics that are in contrast to simple Boolean correct/faulty verdicts like we derive them for MBD. These quantitative semantics allow us thus to capture how well some observed behavior satisfies a property (*SD* in our case), or how badly it failed to do so. To this end, a robustness degree is computed that encodes quantitatively *to which degree* some behavior satisfies or violates a specified STL formula/model. Please note that in run-time verification this means that we investigate a finite prefix that has been observed up to now, and take all its potential continuations into account.

Translating such a concept of quantitative semantics to MBD would be a very interesting extension. This would allow us to judge not only that things are OK, but also the degree to which we should feel safe right now – probably limited to the scope of behavior that is similar to the previously inspected one. It would be only natural to exploit such information in a sensible ranking of diagnoses when deriving inspection strategies based on the diagnoses (complementing the consideration of cardinality and other metrics). For intelligent systems that exploit diagnostic information for achieving operational resilience to disturbances, such quantitative information could also help to assess the severity of issues. This would allow us to extend previous work like [67], where a reliability measure for an agent's action was computed via a data-driven diagnosis approach and then exploited in a monitoring and (re-)planning approach for controlling the agent. Also more elaborate approaches that incorporate MBD and prognosis concepts for achieving operational resilience [25] could profit from this extension to the original MBD theory.

- A Fine-Grained Notion of Consistency for MBD That Supports The Exploitation of Component Degradation
- Resilient Autonomy Supported by Continuous Tracking of Component Degradation via Model-Based Diagnosis

3.5 Models are Approximations

Model-Based Diagnosis is based on having access to an accurate and complete model of the system. If some phenomenon is not modeled, then MBD will have great difficulty pinpointing the ultimate cause for a fault originating from this phenomenon. Let's illustrate this in the case of an overheating resistor R being physically located near a capacitor C on a printed circuit board (PCB) with very minor airflow (like in an enclosure without active cooling). When the heat causes the capacitor C to ultimately fail, MBD will probably pinpoint the capacitor as the culprit – leading to a replacement of the capacitor C but not the resistor R which was the initial root cause. Consequently, R will continue to heat the new C, which will then ultimately fail again (see also our section on fault propagation). If the model had contained a output heat-port for resistor R and an input heat-port for capacitor C, that would only solve one aspect of this challenge. R and C may be very distant from each other on the schematic and only coincidentally be nearby physically, so that the heat model would

need to be based on the spatial and physical details of the assembled PCB. In order to address this case, the system model has to include at least: (1) schematic, (2) heat ports on component models, (3) physical distances between actual components.

There are a multitude of scenarios that we can imagine and where the model is limited in its capabilities for the task at hand – due to one or the other reason. Be it a missing (logical or physical) system dependency, some unmodeled environmental input (e.g., some sort of radiation), an inappropriate abstraction layer, a missed change in the environment, or simply a wrong (hidden) assumption. Currently, MBD research offers no concrete means for assessing and expressing our confidence in a model.

Indeed, there is only initial research related to parts of the challenge, like considering novelty in the context of anomaly and fault detection [30], learning models for diagnostic purposes [32] and assessing our confidence in them during the learning process, as well as techniques for diagnosing formal models [40] that can be integrated into workflows for developing models [43]. To the best of our knowledge there is, however, no comprehensive work investigating the challenge as a whole, i.e, which would allow us to continuously assess and express the quality of an MBD model in general, its suitability for a specific diagnosis problem at hand, as well as the confidence we should have in the results computed for a specific diagnosis problem.

In the future, we hope to see MBD evolving such that it offers us the means to assess the individual aspects of a diagnosis model and diagnosis process qualitatively and quantitatively in terms of quality and confidence. This would allow us not only to progress from the intuition that MBD is sound and complete with respect to the model, but it would allow us also to employ further means for verifying the diagnostic hypotheses, i.e., the checks for consistency between *OBS* and *SD*. That is, using a concept of simulation and exploring the space for free parameters, we could come up with a verdict and confidence estimation for this check in situations where the usual mathematically provable yes/no answer is not possible, e.g., due to the lack of a reasoner that is capable of dealing with weak fault models.

- An Investigation of Measures for Assessing the Quality of MBD Models, Their Correlation to MBD Performance in Practice, and Their Application in Hierarchical MBD
- Exploring the Design Space of MBD Models Based on a Quality Measure for Diagnosis Results
- Assessment-Driven Scaling of the Abstraction in MBD Models and its Connection to Fault Mitigation Capabilities in Resilient Autonomous Systems

3.6 Defining Resilience

A resilient system is one that is robust with respect to faults, physical attacks, cyber-attacks, weather and wear. MBD promises to be a key technology to achieve resilience, because it can be embedded in the system itself so that it allows the system to diagnose a situation and respond to disturbances never anticipated by the designer. This type of active resilience provides a dramatic opportunity to improve operational resilience. The MBD DX community has struggled to find a good a definition of resilience, hence it is challenging to measure the improvement in resilience provided by active resilience. Nevertheless, the resilience of factories, systems, supply chains, etc. is an important topic today and MBD can play a key role in achieving it. That is, it provides diagnostic information for creating the awareness that is needed to make the specific connection between component behavior and the correct, desired overall function.

While resilience is not a new topic, its connection to MBD and the development of an integrated theoretical framework have received insufficient attention. At the Dagstuhl Seminar 24031 *Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis*, there

were promising discussions of these aspects (including the formulation of an appropriate notion of resilience) which we hope come to fruition in the near future. Especially in the context of the suggested PhD topics, we would like to pinpoint the interested reader to the opportunities provided by generative MBD as discussed in Sec. 3.13.

- Achieving System Resilience by Exploiting MBD to Generate the Situational Awareness Necessary for Intelligent Control
- A Notion and Measure of Resilience for Continuously Assessing a System's Resilience Capabilities and Performance
- Supporting Operational Resilience at Design Time: Maximize a System's Resilience Capabilities by Adding Appropriate Run-Time Capabilities Based on Autonomous Agency and Model-Based Diagnosis
- An Architecture for a Resilient Agent: Integrating Learning, MBD, Prognosis and Planning for Achieving Resilient Intelligent Control

3.7 Attacks or Failure?

One main motivation of model-based diagnosis is to diagnose the root causes of a failing system, i.e., derive explanations for some observed unexpected behavior that pinpoint to those system parts that are responsible for the problem by exhibiting faulty behavior. When employing MBD, we often make the tacit assumption that it is indeed a fault that is responsible which simply originates from the properties of the natural world. In general, however, the reason for some system part not working as modeled can be many. One such reason is suggested by the hidden MBD assumption that the system description *SD* is per definition correct in terms of the nominal behavior, i.e., we could simply suffer from a fault in the model itself.

As we discussed briefly in Section 3.5, a model and its interaction with the environment are approximations, so that our understanding modeled in *SD* might be incomplete. Or our understanding is complete, but we made a mistake when formally expressing the system dynamics in *SD*. Consequently, we might not experience a system fault leading to a system failure, but suffer from a fault in the model itself that tells us something went wrong (while actually nothing went wrong). When thinking about resilient autonomous systems, such problems can however manifest in undesired system behavior, the useless expenditure of additional resources, or the activation of redundant system parts that contain faults themselves – so that adversarial attacks are feasible.

While such faults can obviously occur also naturally, it is not difficult to fathom a situation where somebody injected a fault on purpose, such as to be able to trigger specific behavior in a certain context. A corresponding adversarial attack could then lead to either passively or actively triggered malignant behavior – an actively triggered scenario requiring the adversary to have the means to actively trigger the injected behavior via providing specific inputs. Such scenarios received considerable attention in image classification [55], for instance. There are multiple options for providing required inputs to a system and implementing appropriate security countermeasures, such as at the sensor level [3]. We would have to consider, however, also scenarios where an attacker simply focuses on a specific sensor at the physical level by blinding parts of a camera's view or by heating up a temperature sensor. While a formal and diagnostic analysis of the model itself might unveil malignant functionality (depending on the modeling style, the detail level, the level of redundancies in the model, as well as the properties we're investigating), identifying such a physical attack could be quite a challenge - depending on the data and techniques available.

Some faults in the model itself could be identified by diagnosing the model, as, e.g., suggested in [40] where the authors diagnose formal properties in the context of example behavior. Tools like RAT, which support the development of formal models through specific workflows, enable us to explore the semantics of models under development [43]. When combined with diagnostic techniques [40], testing methods such as combinatorial testing [46], or technologies developed to address modeling issues (as discussed in Section 3.5), these tools could allow us to identify and explore potential model problems more effectively.

So we could investigate the consistency and integrity in all the available sensor/input information via concepts that investigate either concrete, analytical dependencies, or alternatives that take advantage of virtual sensing methods. The latter, in particular, would employ ML/DL techniques to learn approximations of specific signals from other (sensor) data, which means that we would *learn* approximations of physical and other dependencies between observed quantities rather than having to model them. Either variant would not only enable us to potentially detect faults in sensors, but would provide us also with more context for the task of differentiating between system faults and issues triggered by attacks.

Currently, MBD does not investigate the reasons behind a diagnosis in detail, in that it focuses on whether the provided diagnosis is *consistent* with the observed behavior – but without evaluating the circumstances or their likelihood (at least most of the time). With the ever-increasing system complexity and the importance of autonomy as a central system feature, we argue that will not be enough though. We must consider diagnoses not only as a means to explain a system's failure (in meeting the expectations covered in *SD*), but we need to go one step further and investigate as well as explain diagnoses themselves. This information will provide the necessary background for resilient systems to autonomously react to faults and other issues.

- A Framework for Developing MBD Models With Enhanced Integrity
- Distinguishing Between Faults and Attacks in Model-Based Diagnosis
- Exploiting Virtual Sensing in MBD by Learning Inherent Dependencies for Assessing the Integrity of Diagnostic Computations
- Explaining Diagnoses Resulting from MBD

3.8 Physical Reasoning

Bearings are a weak point in many physical systems. One failure path is that microscopic metal particles released from wear over time get into the races. The race then becomes pitted and wears prematurely. This is in turn increases the clearances between the race and the cage. Eventually the cage winds up carrying some of the load. But the cage is not designed to carry a load, and collapses, producing catastrophic bearing failure and seizure. The prior sentences describe a common-sense model-based approach that people do all the time. But no current MBD approach can perform this type of reasoning.

Human diagnosticians are very good at this type of reasoning. They think through the behavior physical properties of materials, gases, liquids, stresses, heat, etc. For example, the problems with the heat shields on the space shuttle. Modeling all this physics up front is impossible currently. Instead, human diagnosticians bring their physical knowledge to bear when faced with a difficult diagnostic scenario.

Obviously there are tight connections between this challenge and previous ones like, e.g., the challenge of models being approximations only, or those of capturing fault propagation or component degradation. However, the *SD* here is impossible to capture up front, but is elaborated as diagnostic reasoning proceeds. Essentially the appropriate *SD* is constructed as needed from underlying physics of the system being diagnosed.

- Analyzing the Imperfections of MBD Models of Cyber-physical Systems and Their Effects on the Quality of a Diagnosis Process From a Physics-Centered Point-of-view
- Defining and Exploiting A Measure of Confidence in Modeling Physical Processes for Model-based Diagnosis Purposes
- On-Demand Physics-based Common Sense Reasoning for Novel Diagnostic Scenarios
- Introducing Dynamic Abstraction Into the Diagnostic Process

3.9 Synergies

Many systems in our civilization are massively replicated and widely used: Cars, screws, transistors, airplanes, Xerox copiers, etc. It is extremely inefficient for technicians to constantly rediscover the same fault. When a new fault is discovered in one airplane, we would like that knowledge to communicated immediately to the entire fleet. Otherwise the fault will have to be discovered over and over again.

It is not only the reporting of identified design faults that we allows us to profit from synergies. That is, if we debug some faulty scenario autonomously or manually, taking live or stored data from multiple system copies into account might improve the quality of our debugging process. A motivating example would be when we saw that an airplane's autopilot reacted almost always correctly in a certain situation but failed only once, we would intuitively prioritize explanations that are local to the scenario in which it failed. If it failed more than once, we would probably assume a systematic problem in the design, so that we would use the data observed for all airplanes and the problematic scenario to the end of isolating the relevant parameters and mitigating the issue efficiently in a design repair. Relevant applications would be production robots, autonomous and classic cars, drones, planes and many others.

In a scenario with multiple heterogeneous but collaborating agents, synergies generated by inspecting shared data might allow us to distinguish between isolated sensing faults (like when only one agent reports a temperature deviation) and other problems that are spatial to an agent in space or time from issues that concern all agents. The latter might be changes in hidden assumptions, in the environment, or simply unanticipated and thus unmodeled events and dependencies.

An extreme case of illustrating the potential of synergies is a scenario where we would exploit a digital twin (with the obvious challenge of trying to create a scenario as similar to the actual one as possible) for the purpose of being able to actively gather information for hypothesis discrimination. Simulations with the digital twin could allow us to isolate the right diagnosis or at least shrink the solution space significantly (see Sec. 3.11). We see significant potential for exploiting synergies in diagnostic reasoning, particularly in two types of scenarios. First, in autonomous reactive systems with strong safety requirements, such as autonomous transportation in open environments or military applications. Second, in missions with little room for failure due to hazardous environments and high costs of system loss, like space probe applications. Future research on this challenge could investigate it in the context of answering the questions of

- how to do it, i.e., investigating algorithmic options – considering also security and privacy
- when to do it, i.e., assessing the spatial and temporal locality of information, offering an assessment when – and probably also in the context of which diagnosis – to take potential synergies resulting from non-local information into account
- how to adjust local and fleet-wide likelihoods and assess the resulting diagnosis quality
- Approaches to fleet wide diagnostics learning

3.10 Parallelized MBD

Today, even embedded devices like our smartphones feature multiple cores, and mainstream x86 processors from the consumer market from AMD or Intel offer 12 physical cores capable of handling 24 threads[1] at a 250 € price point. When taking also memory prices into account[2], we can easily see that the hardware for exploiting parallel computation has indeed arrived even in low-cost consumer PCs and wearables. Now, while research on distributed diagnosis [50, 63] offer some potential hints at dividing a diagnosis problem into individually solvable chunks, and naive as well as inefficient algorithms for computing multi-scenario diagnoses [48] offer straightforward options for parallelization, it is quite surprising to observe that we have not yet seen much research dedicated to parallel MBD algorithms, with the notable exception of efforts like [21].

Such algorithms could reduce the experienced wall clock time significantly though and thus directly enhance a major aspect of the MBD user experience – by cutting the time we have to wait for diagnoses. For autonomous system, this is a very welcome effect as well, since this would foster much quicker responses to a problematic situation. We thus argue that research on parallel MBD has the potential to significantly boost MBD's utility. If such research would focus furthermore on dynamically scalable approaches that take the currently available cores, power, and memory into account, MBD would be available also to resource-limited autonomous embedded systems which are the backbone of the current age of digitalization and ubiquitous computing.

The most obvious goal would be to investigate options for a multi-threaded approach, i.e., how to leverage the multiple cores for individual tasks arising in MBD computations. That is, we should aim to investigate multi-threading options in terms of single algorithms, multiple stages, and also at the individual task level – like multi-threaded SMT[3] oracles for the consistency check. Complementing these efforts, research on intelligent concepts for sharing information, results, and memory between the threads could significantly optimize the potential (synchronization) overhead introduced with parallel concepts. It would allow us in particular to avoid unnecessary computations (e.g., computing the same conflict more than once) and also loading the same data multiple times from a disk into memory. A non-negligible side effect would be a minimized resource footprint that is highly welcome for reactive and embedded cyber-physical systems like wearables.

Complementing the exploration of parallelization options, it would be very interesting to investigate architectures of dedicated computation hardware for accelerating MBD. For other applications and tasks, e.g., audio and image processing, generic sub-symbolic AI and cryptography, today's processors feature a multitude of special purpose accelerators and architectural choices. We hope that this would be the case in the future also for MBD, potentially exploiting special structured models as suggested in [10].

- Parallelized MBD Computations for Single- and Multi-Scenario Diagnosis Problems
- A Threaded SMT Engine for Diagnostic Purposes
- A Multi-Core Hardware Architecture for Accelerating MBD Computations

[1] AMD Ryzen 5900XT 12 cores/24 threads/4.8 Ghz, `https://geizhals.at`: 15 offers 240-260€ 02/08/24
[2] a 32GB major brand DDR4 3600Mhz DIMM being available at 60-70€, `https://geizhals.at`, 02/08/24
[3] satisfiability modulo theories

3.11 Improving on the Diagnostic Information Provided by MBD

By default, MBD provides us with a set of diagnoses that are sound and complete with respect to the *SD* and *OBS* [51, 12]. Each such diagnosis suggests a set of components (or knowledge blocks) that, if we assume them to be faulty, explain the observed behavior. In the context of debugging, i.e., when we react on the obtained diagnoses, we basically have two options. That is, we either start probing for new data towards the goal of being able to narrow down the number of diagnoses, or we start investigating the components indicated by the diagnoses – ruling out all those diagnoses that contain a component we find to be correct.

Research on sequential diagnosis [53] and probing strategies [20, 19] has been focusing on the former option, i.e., at determining new measurements or distinguishing sequences [4, 62] that should provide us with more and ideally enough information for discerning what actually happened in the system [37]. It must be noted though that, in general, acquiring additional data does not necessarily entail a monotonically decreasing set of diagnoses. Rather, it might be the case that these new data unveils new problems in the system, such that MBD is non-monotonic by default [51, 12].

Irrespective of whether we employ such means for narrowing down the number of diagnoses, we will have to start inspecting the system based on one or more diagnoses at some point. When doing so, we would certainly appreciate any situational data that would accompany the diagnoses. Such data could help in explaining what makes a diagnosis a diagnosis, and would assist us consequently in investigating the component by telling us where to look in the component. This is precious information since the assumption of perfect bug understanding, i.e., the assumption that we can easily isolate the fault in a component, requires extensive expertise with a system and correlates seldomly with reality.

Research on this usability aspect of MBD has been neglected though. This is interesting insofar as the conflicts obtained in the diagnosis process contain such information – like timing information when a fault supposedly occurred in sequential behavior [40]. In MBD algorithms like HS-DAG, GDE or RC-Tree, we exploit, however, only the part of a conflict associated with health state variables/abnormal predicates. The additional explanatory information that was already computed and would be appreciated during debugging, is ignored algorithmically.

One could also imagine to explore further justifications for a conflict/diagnosis (possibly to the extent of offering a complete set) and to provide a user with this valuable diagnostic information that even comes with some guarantees (depending on the exhaustiveness of the exploration) and touches on the ideas behind truth maintenance systems. Research on chronicles and chronicle-based diagnosis [23] considers such justifications, but for serving an entirely different purpose, i.e., that of computing a diagnosis via searching in *OBS* for relevant patterns. We suggest to rather aim at isolating diagnosis problem-specific justifications for the derived diagnoses, and to exploit these data for debugging – and potentially also for deriving probing strategies.

- An Augmented MBD Interface for Enhancing the Effectiveness of Debugging Reactive Stateful Systems
- A Comparison of Augmented User Interfaces for MBD From the Perspective of Automated and Human Debugging

3.12 Where Does the Model Come From?

MBD critically depends on the quality of the model (*SD*) used in the reasoning process. That is, we have to model all the aspects that we want to be taken into account in the reasoning process. As we discussed in Sec. 3.5, a model is always an approximation and it is only

as good as we make it (cf. our discussion about wear in Section 3.4, or about attacks and failures in Section 3.7). To this end, we have to aim for an approximation level that allows us to achieve the performance in terms of diagnosability and quality that we're aiming for. In medical diagnosis this relates to epistemic uncertainty (vs. aleatoric uncertainty).

The question is now how we can achieve that, and which type of knowledge we should include in a model. The unsatisfying answer is that it is impossible to provide a universal answer since MBD is completely domain- and task-agnostic from a technological perspective. While this in turn means that we can't come up with universal rules of thumb, we could partly answer this question if there was a notion of confidence in a model. For this we would need to be able to assess a model's quality in the context of the application domain as well as the diagnostic questions we would like to answer.

Aside the consideration of quality aspects, we have take also into account that WFM and SFM models differ to some degree – in that a WFM model leaves a component's behavior completely undefined for the faulty case. Consequently, employing a WFM model does not require us to define all the faulty behavior variants (which we might not be able to do), but it also restricts us in terms of the engines that we can use for the consistency check. For example, we can't use a simulation model for WFM MBD, but it can serve in combination with a test oracle (the latter for assessing the outputs) for SFM MBD if all inputs are known. For SFM MBD, on the other hand, the computations are more complex (see Sec. 2) and we need to obtain and model a complete, representative list of faults and their behavioral descriptions. So there is a trade-off that we need to make between the engines we can use, the information that we require to be available, the efforts we have to spend on the modeling, the preciseness of the results, and the complexity of the reasoning process.

Regardless of a corresponding decision and an application's details, we would certainly appreciate (semi-) automated assistance when we have to create an MBD model. Especially for abductive diagnosis there are indeed concepts available that help a user in generating the appropriate models [44, 45]. With CatIO [33] there is also tool support that allows a user to generate an abductive or an MBD model. As a side-note we would like to point out that SFM MBD models can be indeed very similar or equivalent to abductive models, depending on the abstraction level used and whether we have models of individual components or just encode the effect of some faults on the outputs on a qualitative level.

In addition to workflow assistance for model creation, the challenge of creating MBD models as a symbolic AI technique can greatly benefit from sub-symbolic AI approaches that facilitate model learning. Automata learning [38], for instance, allows us to employ MBD by learning a formal model that can be easily adopted for MBD [32] – supported by tools like AALpy [31]. Virtual sensing concepts (see Sec. 3.7) can be employed to augment existing models with additional signals that approximate certain interconnections and dependencies in a system (or learn aspects from real data that were or could not be modeled). This would support us in recognizing model limitations and issues by comparing the results of different virtual sensors for the same signal or a virtual signal with a real one. There is an abundance of further research on learning models [60, 5, 29] that contributes to tackling some of the aspects of this challenge, but we are to the best of our knowledge not aware of a comprehensive technology, methodology and framework for developing (and assessing) MBD models.

We argue that modeling workflows from other fields, such as those used in developing hardware specifications and models [43], should be adapted for MBD model development. These workflows, and their underlying technologies that help users assess and verify model semantics and quality (as implemented in tools like RAT [8]), could significantly improve the

MBD modeling process. Considering the above mentioned tools, we can, for instance, achieve this easily for models that use LTL-like concepts – like finite state machines or similar logics – due to MBD having been translated already to LTL [40] (sharing even a code-base with technology for generating test oracles [47]).

We do need such technology for a multitude of modeling formalisms though, and we thus require a universal concept and technology base for developing corresponding frameworks. This would allow us to translate technologies like diagnosability assessment [7, 64, 57], design exploration via simulation [43], design assessment via verification [43], diagnostic support during the development [40], and future research published at dedicated conferences like ACM/IEEE MODELS[4], DAC[5] or DATE[6] (in order to name just a view) more easily.

- Formal Analysis of MBD Models for Quality Assessment and Development Support
- A Framework for the Development of MBD Models for Cyber-Physical Systems
- A Formal Approach to Maintaining MBD Models via (Semi-)Automated Correction Triggered by Novelty and Change Detection
- A Framework for a Structured Exploration of the Design Space for MBD Models

3.13 MBD as a design tool, or "Redesign by MBD"

MBD has a long tradition in being used to isolate the root causes of a problem – the purpose for which it was developed. We would like to point out, though, that we could employ MBD also in a generative way. Let us consider that for SFM MBD, we move from answers suggesting that *this set of faulty components explains the problem* to stating that *if these components show this or that alternative behavior, this would match OBS*. As already exploited in [40], a tiny switch in context now unveils that MBD provides us also with the means to suggest repairs – like for the design of a system. That is, we can rethink fault modes as alternative behaviors and then determine a set of changes to a system model (employing alternative behavior for some components) such that *SD* would be consistent again with some supposed witness from a requirements document, i.e., with a trace that describes valid behavior. The same is true for behavior that was described in the requirements document as counterexample, but where we recognize in our design process that it is actually allowed by the system. A similar application domain would that be of planning, i.e., tasks where we seek to reach a desired goal via executing a certain sequence of actions.

This extension to MBD's scope is not entirely new, so that it was exploited for repairing LTL models [40], repairing logical models and interconnections [15] and generating designs via configuring universal components [17]. Technologically, this task is close to solving configuration problems. In practice, it is however the number and complexity of the component's alternative behavior models (which relate to mutation operators as used in mutation testing and fault injection) that determine the effectiveness and performance of a corresponding approach. In [40], for instance, those operators were quite limited in terms of their number and complexity – which is supported by the common (sometimes hidden) assumptions behind the competent programmer hypothesis [2] and an opportunistic reverse view on the coupling effect [39]. These assumptions however not always correlate with reality, and especially so in early design stages. We thus argue that these works were initial steps, and that we need future research on how to efficiently explore the repair/design space in order to fully exploit MBD's generative capabilities.

[4] http://www.modelsconference.org
[5] https://www.dac.com/
[6] https://www.date-conference.com/

It is obvious that the more complex and numerous the mutation operators become, the more optimal a solution we might find – at the cost of a having to conquer an immensely growing search space. For is assessment, we can establish a simple upper bound $O(n^m)$ with m the number of components and n the maximum number of mutation options per component (cf. SFM MBD). For this estimation, we inherently assume mutual exclusiveness of the mutations per component though, and also completely ignore the option of having mutation sequences.

The envisaged problem is thus of very high complexity, and we argue that there is a need for sophisticated solutions that will allow us to conquer the complexity in acceptable time and with acceptable resources. An effective approach would have tremendous potential though and could generate a huge impact. So it would certainly be a most valuable asset for designing, maintaining, and also modeling systems considering the discussion of the previous challenge. The largest impact we see, however, for resilient autonomous reactive systems, i.e., for systems that are allowed to evolve themselves and their behavior in order to adapt to changes in the environment, themselves (degradation), circumstances (regulations), or their mission (like when being required to tackle new tasks). For such systems, we could then add an automated designer to the system that re-designs it at run-time as required for living up dynamic changes as we experience them in an open world.

- Efficient Exploration Strategies for Conquering the Search Space in Generative MBD
- A Generative MBD Framework for Design Space Exploration and Repair
- Achieving Operational Resilience via Adding an MBD-based Run-Time Designer to Reactive Systems

3.14 Fusing Symbolic and Sub-symbolic AI in the context of MBD

In many of the previous sections, we discussed options to fuse symbolic and sub-symbolic techniques for improving MBD. This included sub-symbolic learning of symbolic MBD models, sub-symbolic virtual sensing as a means for assessing the quality of symbolic MBD models, but we could also learn a symbolic model from a neural network to employ MBD for assessing encountered generalization problems [32].

In the literature, we can also find classifiers for diagnosis [26], medical diagnosis based on image classification [65], data-driven approaches like spectrum-based fault localization (SFL) [1] and many other sub-symbolic approaches that complement and compete with MBD. These approaches often allow a much quicker inference/computation of diagnoses compared to MBD, since we move the complexity from the run-time to the training phase, or are content with an approximation as used for SFL. There are also downsides, like that we have to spend a lot of resources when (re-)training classifiers whenever a system changes. The limitations of the trained neural networks are also less graspable from an analytical point of view, i.e., when trying to isolate for which situations we will encounter problems. Furthermore, we have usually little fault data available for training purposes.

As visible from our discussions of the previous challenges, we argue that combining symbolic and sub-symbolic techniques will allow us to leverage the advantages of both approaches in tackling our challenges. While hybrid AI and the fusion of symbolic and sub-symbolic AI concepts are indeed hot topics and are gaining in attention in general, we need to increase our corresponding research efforts also specifically in the field of diagnosis and MBD. Important discussions in this direction were led at the Dagstuhl Seminar 24031 *Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis*, and we

hope to see a lot of upcoming research. Just as diagnosis research in the control community was bridged with diagnosis research in the AI community [58], we need to build similar bridges between the sub-symbolic and symbolic worlds in AI.

- A Survey and Evaluation of Approaches Bridging Symbolic with Sub-Symbolic AI in the Context of MBD

4 Summary

In this manuscript, we showed that Model-Based Diagnosis has never lost in attractiveness, nor in its relevance. There is no doubt that we are still in need of approaches like MBD that allow us to structurally reason about encountered problems, and especially so when taking the ever-increasing complexity of the tasks we are facing in our everyday lives into account. For being able to efficiently and effectively solve the most complex diagnostic tasks, there is, however, an abundance of research questions left that we still need to answer – even after 40 years of research on MBD.

We discussed a corresponding set of MBD challenges that we think are important to address, and which are related to a variety of MBD aspects. We covered the importance and relevance of each challenge in brief discussions, and suggested potential topics for PhD theses that could propel the state-of-the-art in MBD research significantly. While we discussed the challenges individually (hinting at connections every now and then), it is important to point out that there are a lot of cross-connections in terms of motivating agendas – which would in turn multiply the suggested PhD topics from a technological perspective.

In our conclusion, we would like to state again that we were certainly not able to include all the important challenges in this discussion (there is a space limit), and that we would be very grateful for feedback from the interested reader that we could use to improve our discussion for future manuscript versions. As a concluding remark, we would like to express our sincere hopes that this paper would serve as motivation for one or the other researcher and PhD student to work on tackling the described challenges and to contribute to the Model-Based Diagnosis technology of tomorrow.

References

1 R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques*, pages 89–98, 2007. doi:10.1109/TAIC.PART.2007.13.

2 A. T. Acree, T. A. Budd, R. A. De-Millo, R. J. Lipton, and F. G. Sayward. Mutation Analysis, Technical Report GIT-ICS-79/08. Technical report, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, 1979.

3 C. M. Ahmed, A. P. Mathur, and M. Ochoa. NoiSense Print: Detecting Data Integrity Attacks on Sensor Measurements Using Hardware-based Fingerprints. *ACM Transactions on Privacy and Security*, 24(1), September 2020. doi:10.1145/3410447.

4 R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 363–372, 1995. doi:10.1145/225058.225161.

5 J. L. Augustin and O. Niggemann. Graph Structural Residuals: A Learning Approach to Diagnosis, 2023. arXiv:2308.06961, doi:10.48550/arXiv.2308.06961.

6 D. W. Binkley and K. B. Gallagher. Program Slicing. *Advances in Computers*, 43:1–50, 1996. doi:10.1016/S0065-2458(08)60641-5.

7 S. Biswas, D. Sarkar, S. Mukhopadhyay, and A. Patra. Diagnosability Analysis of Real Time Hybrid Systems. In *IEEE Int. Conf. on Industrial Technology*, pages 104–109, 2006.

8 R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A Tool for the Formal Analysis of Requirements. In *19th Int. Conf. on Computer Aided Verif.*, pages 263–267, 2007.

9 C. Coquand, Y. Pencolé, and A. Subias. Diagnosabilization of Time Petri net for timed fault. *IFAC-PapersOnLine*, 56(2):8648–8653, 2023. 22nd IFAC World Congress.

10 A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research (JAIR)*, 8(1):165–222, June 1998. doi:10.1613/JAIR.462.

11 J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing Diagnoses and Systems. *Artificial Intelligence*, 56(2-3):197–222, 1992. doi:10.1016/0004-3702(92)90027-U.

12 J. de Kleer and B. C. Williams. Reasoning about Multiple Faults. In *5th National Conf. on Artificial Intelligence Volume 1: Science*, pages 132–139, 1986.

13 J. de Kleer and B. C. Williams. Diagnosis with Behavioral Modes. In *11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1330, 1989.

14 Johan de Kleer. Hitting set algorithms for model-based diagnosis. In *22nd International Workshop on Principles of Diagnosis (DX'11)*, 2011.

15 Johan de Kleer, Alexander Feldman, and Ion Matei. The duality of design and diagnosis. In *29th Int. Workshop on Principles of Diagnosis (DX'18), Warsaw, Poland*, 2018. URL: http://dekleer.org/Publications/ijcai2019r.pdf.

16 A. Donzé, T. Ferrère, and O. Maler. Efficient Robust Monitoring for STL. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, pages 264–279, 2013.

17 A. Feldman, J. de Kleer, and I. Matei. Design Space Exploration as Quantified Satisfaction, 2021. arXiv:1905.02303.

18 R. Greiner, B. A. Smith, and R. W. Wilkerson. A Correction to the Algorithm in Reiter's Theory of Diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989. doi:10.1016/0004-3702(89)90079-9.

19 B. Han, S.-J. Lee, and H.-T. Yang. Comments on the theory of measurement in diagnosis from first principles. *Information Sciences*, 121(3):349–365, 1999. doi:10.1016/S0020-0255(99)00034-1.

20 A. Hou. A theory of measurement in diagnosis from first principles. *Artificial Intelligence*, 65(2):281–328, February 1994. doi:10.1016/0004-3702(94)90019-1.

21 D. Jannach, T. Schmitz, and K. Shchekotykhin. Parallelized hitting set computation for model-based diagnosis. In *29th AAAI Conf. on Artificial Intelligence*, pages 1503–1510, 2015.

22 U. Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *19th National Conf. on Artifical Intelligence (AAAI)*, pages 167–172, 2004.

23 X. Le Guillou, M.-O. Cordier, S. Robin, and L. Rozé. Chronicles for On-line Diagnosis of Distributed Systems. In *18th European Conf. on Artificial Intelligence*, pages 194–198, 2008.

24 C. Li, D. Chen, X. Liu, M. Shahidehpour, H. Yang, H. Liu, W. Huang, J. Wang, X. Deng, and Q. Zhang. Fault mitigation mechanism to pave the way to accommodate over 90% renewable energy in electric power systems. *Applied Energy*, 359:122623, 2024.

25 I. Matei, W. Piotrowski, A. Perez, J. de Kleer, J. Tierno, W. Mungovan, and V. Turnewitsch. System Resilience through Health Monitoring and Reconfiguration. *ACM Transactions on Cyber-Physical Systems*, 8(1), January 2024. doi:10.1145/3631612.

26 I. Matei, M. Zhenirovskyy, J. de Kleer, and A. Feldman. Classification-based Diagnosis Using Synthetic Data from Uncertain Models. *Annual Conference of the PHM Society*, 10(1), 2018.

27 A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling Model-Based Diagnosis to Boolean Satisfaction. In *26th AAAI Conference on Artificial Intelligence*, pages 793–799, 2012.

28 Raj Minhas, Johan De Kleer, Ion Matei, Bhaskar Saha, Bill Janssen, Daniel G Bobrow, and Tolga Kurtoglu. Using fault augmented modelica models for diagnostics. In *Proceedings of the 10th international modelica conference*, pages 437–445, 2014.

29 L. Moddemann, H. Steude, A. Diedrich, I. Pill, and O. Niggemann. Extracting Knowledge using Machine Learning for Anomaly Detection and Root-Cause Diagnosis. In *29th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2024. to appear.

30 S. Mohan, W. Piotrowski, R. Stern, S. Grover, S. Kim, J. Le, Y. Sher, and J. de Kleer. A domain-independent agent architecture for adaptive operation in evolving open worlds. *Artificial Intelligence*, 334:104–161, 2024.

31 E. Muškardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler. AALpy: an active automata learning library. *Innovations in Systems and Software Engineering*, 18(3):417–426, 2022. doi:10.1007/S11334-022-00449-3.

32 E. Muškardin, I. Pill, M. Tappler, and B. Aichernig. Automata Learning Enabling Model-Based Diagnosis. In *32nd Int. Workshop on Principle of Diagnosis (DX)*, September 2021. URL: https://www.hsu-hh.de/imb/en/dx-2021.

33 E. Muškardin, I. Pill, and F. Wotawa. CatIO - A Framework for Model-Based Diagnosis of Cyber-Physical Systems. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, and Z. W. Raś, editors, *Foundations of Intelligent Systems*, pages 267–276, 2020.

34 S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(3):348–361, 2007. doi:10.1109/TSMCA.2007.893487.

35 I. Nica, I. Pill, T. Quaritsch, and F. Wotawa. The Route to Success - A Performance Comparison of Diagnosis Algorithms. In *23rd International Joint Conference on Artificial Intelligence*, pages 1039–1045, 2013.

36 I.-D. Nica and F. Wotawa. ConDiag - Computing minimal diagnoses using a constraint solver. In *23rd International Workshop on Principles of Diagnosis*, 2012.

37 M. Nica, S. A. Nica, and F. Wotawa. Using Distinguishing Tests to Reduce the Number of Fault Candidates. In *21st International Workshop on the Principles of Diagnosis*, 2010.

38 O. Niggemann, B. Stein, A. Maier, A. Vodenčarevič, and H. Kleine Büning. Learning behavior models for hybrid timed systems. In *26th AAAI Conf. on Art. Intell.*, pages 1083–1090, 2012.

39 A. Offutt. The Coupling Effect: Fact or Fiction? *SIGSOFT Software Engineering Notes*, 14(8):131–140, November 1989. doi:10.1145/75309.75324.

40 I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *23rd Int. Joint Conf. on Artificial Intelligence*, pages 1053–1059, 2013.

41 I. Pill and T. Quaritsch. RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm. In *IEEE Int. Symp. on Software Reliability Engineering Workshops (ISSREW)*, pages 78–84, 2015.

42 I. Pill, T. Quaritsch, and F. Wotawa. Parse tree structure in LTL requirements diagnosis. In *2015 IEEE Int. Symp. on Software Reliability Engineering Workshops*, pages 100–107, 2015.

43 I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *43rd Annual Design Automation Conference*, pages 821–826, 2006.

44 I. Pill and F. Wotawa. Fault Detection and Localization Using Modelica and Abductive Reasoning. *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems*, pages 45–72, 2018. doi:10.1007/978-3-319-74962-4_3.

45 I. Pill and F. Wotawa. On Using an I/O Model for Creating an Abductive Diagnosis Model via Combinatorial Exploration, Fault Injection, and Simulation. In *29th International Workshop on Principles of Diagnosis (DX'18)*, 2018.

46 I. Pill and F. Wotawa. Exploiting observations from combinatorial testing for diagnostic reasoning. In *30th Int. Workshop on Principles of Diagnosis*, 2019.

47 I. Pill and F. Wotawa. Extending Automated FLTL Test Oracles with Diagnostic Support. In *IEEE Int. Symp.on Software Reliability Engineering Workshops*, pages 354–361, 2019.

48 I. Pill and F. Wotawa. Computing Multi-Scenario Diagnoses. In *31st International Workshop on Principles of Diagnosis, DX ; Conference date: 26-09-2020*, 2020. URL: http://dx-2020.org/.

49 A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

50 G. M. Provan. A Model-Based Diagnosis Framework for Distributed Embedded Systems. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and Mary-Anne Williams, editors, *8th Int. Conf. on Principles and Knowledge Representation and Reasoning*, pages 341–352, 2002.

51 R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

52 N. Rhinehart, J. He, C. Packer, M. A. Wright, R. McAllister, J. E. Gonzalez, and S. Levine. Contingencies from Observations: Tractable Contingency Planning with Learned Behavior Models, 2021. arXiv:2104.10558.

53 P. Rodler. On Active Learning Strategies for Sequential Diagnosis. In *28th Int. Workshop on Principles of Diagnosis*, volume 4, pages 264–283, 2018.

54 P. Rodler. Memory-limited model-based diagnosis. *Artificial Intelligence*, 305:103681, 2022. doi:10.1016/J.ARTINT.2022.103681.

55 L. Schwinn, R. Raab, A. Nguyen, D. Zanca, and B. Eskofier. Exploring misclassifications of robust neural networks to enhance adversarial attacks. *Applied Intelligence*, 53(17):19843–19859, 2023. doi:10.1007/S10489-023-04532-5.

56 S. Siddiqi and J. Huang. Hierarchical diagnosis of multiple faults. In *20th Int. Joint Conference on Artifical Intelligence*, pages 581–586, 2007.

57 X. Su, M. Zanella, and A. Grastien. Diagnosability of discrete-event systems with uncertain observations. In *25th Int. Joint Conf. on Artificial Intelligence*, pages 1265–1271, 2016.

58 L. Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27:1–16, 2014. doi:10.1016/J.ENGAPPAI.2013.09.018.

59 J. Weber and F. Wotawa. Diagnosing dependent failures - an extension of consistency-based diagnosis. In *18th Int. Workshop on Principles of Diagnosis (DX-07)*, 2007.

60 A. Windmann, H. Steude, and O. Niggemann. Robustness and Generalization Performance of Deep Learning Models on Cyber-Physical Systems: A Comparative Study, 2023. arXiv:2306.07737, doi:10.48550/arXiv.2306.07737.

61 F. Wotawa. A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001. doi:10.1016/S0020-0190(00)00166-6.

62 F. Wotawa, M. Nica, and B. K. Aichernig. Generating Distinguishing Tests Using the Minion Constraint Solver. In *3rd Int. Conf. on Software Testing, Verification and Validation, Workshops Proceedings*, pages 325–330, 2010. doi:10.1109/ICSTW.2010.11.

63 F. Wotawa and I. Pill. On classification and modeling issues in distributed model-based diagnosis. *AI Communications*, 26(1):133–143, January 2013. doi:10.3233/AIC-2012-0548.

64 L. Ye, P. Dague, D. Longuet, L. B. Briones, and A. Madalinski. Fault manifestability verification for discrete event systems. In *22nd European Conf. on Artificial Intelligence*, pages 1718–1719, 2016. doi:10.3233/978-1-61499-672-9-1718.

65 X. Yu, H. Luo, J. Hu, X. Zhang, Y. Wang, W. Liang, Y. Bei, M. Song, and Z. Feng. Hundredfold Accelerating for Pathological Images Diagnosis and Prognosis through Self-reform Critical Region Focusing. In *33rd Int. Joint Conf. on Artificial Intelligence*, pages 1607–1615, 2024.

66 S. A. Zarghami. Resilience to disruptions: a missing piece of contingency planning in projects. *International Journal of Production Research*, 62(17):6029–6045, 2024. doi:10.1080/00207543.2024.2306474.

67 M. Zimmermann, F. Wotawa, and I. Pill. Pursuing Intelligent Behavior in Cyber-Physical Systems by Lightweight Diagnosis. *Advanced Intelligent Systems*, 4(4), 2022. doi:10.1002/AISY.202100224.

Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research

Daniel Vranješ
Helmut Schmidt University, Hamburg, Germany

Jonas Ehrhardt
Helmut Schmidt University, Hamburg, Germany

René Heesch
Helmut Schmidt University, Hamburg, Germany

Lukas Moddemann
Helmut Schmidt University, Hamburg, Germany

Henrik Sebastian Steude
Helmut Schmidt University, Hamburg, Germany

Oliver Niggemann
Helmut Schmidt University, Hamburg, Germany

Abstract

Machine learning is becoming increasingly important in the diagnosis and planning fields, where data-driven models and algorithms are being employed as alternatives to traditional first-principle approaches. Empirical research plays a fundamental role in the machine learning domain. At the heart of impactful empirical research lies the development of clear research hypotheses, which then shape the design of experiments. The execution of experiments must be carried out with precision to ensure reliable results, followed by statistical analysis to interpret these outcomes. This process is key to either supporting or refuting initial hypotheses. Despite its importance, there is a high variability in research practices across the machine learning community and no uniform understanding of quality criteria for empirical research. To address this gap, we propose a model for the empirical research process, accompanied by guidelines to uphold the validity of empirical research. By embracing these recommendations, greater consistency, enhanced reliability and increased impact can be achieved.

2012 ACM Subject Classification Computing methodologies → Machine learning

Keywords and phrases machine learning, hypothesis design, research design, experimental research, statistical testing, diagnosis, planning

Digital Object Identifier 10.4230/OASIcs.DX.2024.7

Related Version *Previous Version*: https://arxiv.org/abs/2405.18077

1 Introduction

Deductive and abductive reasoning play crucial roles in both theoretical and empirical research. Deductive reasoning, often used in theoretical research, involves deriving specific predictions from general principles or axioms. This approach ensures internal consistency within a logical framework but does not require empirical testing. In contrast, abductive reasoning, which is foundational in empirical research, involves forming plausible hypotheses to explain observations. These hypotheses are then tested through experiments to gather empirical evidence. While deductive reasoning provides a solid foundation for developing theoretical models, abductive reasoning bridges the gap between theory and practice by enabling the formulation and empirical validation of testable hypotheses.



© Daniel Vranješ, Jonas Ehrhardt, René Heesch, Lukas Moddemann, Henrik Sebastian Steude, and Oliver Niggemann;
licensed under Creative Commons License CC-BY 4.0



Theoretical research in machine learning (ML) involves the development and analysis of models and algorithms through mathematical formalisms and proofs, offering insights into their properties, performance guarantees, and limitations. This foundational work is essential for understanding the principles that govern ML systems, guiding the design of new algorithms, and providing a basis for interpreting their behavior. The analytical and formal nature of this approach allows for certain and specific evidence, but is not applicable for most hypotheses.

Empirical research, on the other hand, plays a crucial role in the ML domain through testing and validating of theoretical models and hypotheses in practical settings where theoretical proof is not possible [15].

ML is inherently more empirical and experimental compared to symbolic AI due to its reliance on data-driven methods to develop and refine algorithms. Unlike symbolic AI, which relies on predefined rules and logic, ML systems learn patterns and make decisions based on large datasets. This necessitates extensive experimentation to evaluate the effectiveness, robustness, and generalizability of models across diverse and often unseen data. The iterative process of training, testing, and validating ML models requires empirical evidence to ensure their accuracy and reliability in real-world applications, making experimentation a cornerstone of the ML research methodology.

Despite the existence of abstract process models and guidelines for the operational aspects of ML (MLOps), the scientific ML community lacks standard process models for conducting research and even a common conception of what good research practices are in general and in detail. Empirical ML research faces several critical challenges that can impact its scientific rigor and the applicability of its findings. One significant issue is the problem of falsifiability, where some empirical studies may not clearly delineate conditions under which the proposed models could be proven wrong, leading to ambiguous interpretations of results. Replicability and reproducibility are also major concerns. The former refers to the ability of different researchers to achieve the same results using the same dataset and methodology, while the latter pertains to the capacity to achieve consistent results across different settings, datasets, and experimental configurations. The complexity and variability inherent in ML models, combined with the use of proprietary datasets or software and insufficient documentation, often hinder these aspects. Furthermore, generalizability is a persistent problem, as models trained and tested on specific datasets or in controlled environments may not perform well on unseen data or in real-world applications, questioning the external validity of the research. These challenges underscore the need for rigorous methodological standards, transparent reporting, and comprehensive evaluation metrics in empirical ML research to ensure its findings are robust, reliable, and broadly applicable.

Within this paper we focus on the design and execution of experiments as one encapsulated part of the ML pipeline from a scientific research perspective without any emphasis on the overall aspects of the life cycle of ML software or organizational aspects as e.g., promoted by process models such as CRISP-ML(Q) [22] or the ISO/IEC 23053:2022.

In reviewing diverse sources, it becomes evident that certain terminologies are employed with varying definitions. [6, 18, 9] highlight, for instance, that the concepts of replicability and reproducibility lack uniformity in definition, with a plethora of terms and interpretations in circulation. Given the critical need for a shared lexicon in scholarly discourse, we propose the following definitions to clarify our interpretation of these key terms within this publication.

Scientific empiricism. Scientific empiricism is a philosophical approach that underscores the importance of empirical evidence in the acquisition and validation of knowledge. Within this framework, verificationism and falsification play crucial roles as methodologies for

evaluating scientific theories. Verificationism seeks to confirm the validity of hypotheses by demonstrating that empirical evidence is consistent with them, highlighting the role of positive evidence in supporting scientific claims. On the other hand, falsification, popularized by Karl Popper [15], argues that scientific theories cannot be conclusively verified but can be robustly tested by seeking evidence that could potentially refute them. This approach emphasizes the critical importance of disprovability as a hallmark of scientific knowledge, suggesting that theories must be open to empirical testing and possible rejection. Together, verification and falsification embody the empirical spirit of scientific inquiry, advocating for a rigorous and evidence-based approach to understanding the natural world. Verification and falsification can even be attributed with a methodological symmetry [19]. Research hypotheses define the aim of the research and represent its evaluation criteria. To enable falsifiability or verification, they must be precise and answerable within the scope of the conducted research. Experiments must be conducted in a way to generate data which allows for a statistically significant analysis of hypotheses and empirical evidence shall be used to prove or disprove the hypotheses. Ensuring inter-subjectivity in ML research is crucial, as it allows different researchers to independently reproduce and validate findings, thereby enhancing the reliability and credibility of the results. This collective verification process strengthens the overall robustness and acceptance of empirical research within the scientific community. This step requires the replicability or reproducibility of the research.

Replicability (Repeatability). This principle dictates that identical results should be generated when an experiment is rerun using the same data, code and computational environment (hardware and software). The researchers conducting the study might also be considered as a variable, referring to repeatability when the same team performs the experiment multiple times, and replicability when a different team performs an experiment under exactly the same conditions. Since an ML experiment can be fully defined by the hardware, software environments and code used, we do not distinguish between these cases in this paper. Conducting an experiment multiple times in the same environment should yield consistent results. Since ML algorithms and computational methods may involve a stochastic nature, results may vary slightly across repetitions, leading to non-identical outcomes. However, they remain consistent as long as the impact of these variations is negligible compared to the experimental variables of interest. Replication can be used to detect fraud [2] or to check for stochastic behavior of the program leading to arbitrary results over multiple runs. Here the ML method of interest is strictly bound to its implementation resulting in the evaluation of the concrete ML program. Replicability is the lowest level of independent verification of research through third parties. The replicability within the initial research team resembles only a repeatability, which allows for the evaluation of the stochastic behavior but does not increase validity regarding the approval by the scientific community.

Reproducibility. Reproducibility has a broader understanding of the recreation of results than replicability. In [6] reproducibility is defined as "the ability of an independent research team to produce the same results using the same AI method based on the documentation made by the original research team". Here, the recreation can differ in terms of code or computational environment used for the experiments. Results should still support the hypotheses of the initial research. Using the same experimental methods should lead to similar results in multiple studies. Since this is not bound to a specific implementation, reproducibility does not evaluate a specific ML program but an ML method [6].

(Cross-Domain) Generalizability. In cyber-physical systems (CPS), significant variations in the underlying physical properties across different systems challenge the predictability of algorithm transferability. This variability necessitates careful consideration of the specific application data used in algorithm evaluations. To foster robust and generalizable hypotheses, it is imperative to treat the data or use case as a variable and conduct evaluations across a diverse array of use cases. Such an approach ensures that empirical findings are not confined to a single application, enhancing the broader applicability of the results. Furthermore, in tasks like anomaly detection or classification, assessing the cross-domain generalizability of new algorithms is crucial. This aspect can be viewed as an extreme form of reproducibility, where the algorithms must maintain performance despite significant shifts in data characteristics and context.

2 State of the Art

Researchers from different application domains within the ML community have already (partially) addressed attributes of good research, as well as current problems of and possible solutions for such research.

From a theoretical perspective, the cornerstone of research across various fields is the development of research hypotheses. Central to this endeavor are universally applicable concepts such as inductive and deductive reasoning, along with the principles of verification and falsification [19, 10]. Although no singular methodology reigns supreme, each approach fundamentally aims to craft research hypotheses that are subject to rigorous testing and validation through coherent argumentation.

On the applied side, as elaborated in [2, 3] the majority of the scientific community views the replicability of research as desirable, whilst others do not view the sole replicability as good research and in contrast promote the importance of reproducibility. ML research has significant potential for improvement regarding its replicability as stated in [11], due to a high amount of research being published without code or data.

We randomly select 100 research papers from the 2023 International Joint Conference On Artificial Intelligence (IJCAI). We look at papers #1 to #75 of the main track and papers #378 - #399 of the ML track and evaluate those that contain experimental research for methods applied to the experiments. We find the mentions of the following methods in multiple research papers. No paper has used all the methods. We do not perform a quantitative evaluation of the found methods, since it is likely that authors applied methods that they did not explicitly mention and the evaluation would not represent the real distribution.

Benchmark against baselines. Comparing the proposed algorithm with state-of-the-art competitors is fundamental. This involves identifying current leading algorithms in the field and benchmarking the proposed algorithm's performance against these. The comparison should be fair and transparent, using the same datasets, pre-processing techniques, and evaluation metrics. Hyperparameters shall be optimized for baselines as well. This helps in assessing the true innovation or improvement offered by the proposed algorithm.

Using multiple data sets. Benchmarking on multiple diverse datasets is crucial to demonstrate the robustness and generality of the algorithm. Each dataset should represent different aspects of the problem domain, varying in size, complexity, and inherent biases. This approach ensures that the algorithm's performance is not tailored to the idiosyncrasies of

a single dataset but is broadly applicable across the domain. Where applicable, commonly used benchmarks shall be used for evaluation to allow for comparison with state-of-the-art alternatives.

Data shuffling. Data shuffling is a preprocessing technique used to enhance the generalization ability of models by randomizing the order of data points in the training dataset. This process helps in mitigating the risk of learning spurious patterns that may arise from the order-dependent biases inherent in the data collection process.

By ensuring that each mini-batch of data, used during the training phase, contains a diverse set of examples, shuffling helps in preventing the model from memorizing the sequence of training examples, thereby improving its ability to generalize from the training data to unseen data. In ML tasks involving time series data, data shuffling can be used when the series has been sliced into windows or segments, allowing models to learn from a variety of temporal patterns without compromising the sequential integrity within each segment.

Cross-validations. Cross-validation is a statistical method to assess how the results of an algorithm will generalize to an independent data set. It is primarily used to estimate the skill of a model on unseen data. This technique involves partitioning a sample of data into complementary subsets, performing the learning on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). The process is repeated multiple times, with different partitions, to reduce variability. Common forms of cross-validation are k-fold cross-validation and leave-one-out cross-validation.

Hyperparameter optimization. Hyperparameters control the learning process and can significantly affect performance. Hyperparameter optimization seeks the optimal set of hyperparameters that yields the best performance on a validation set. Techniques such as grid search, random search, or more sophisticated methods like Bayesian optimization can be used. This process ensures that the algorithm's potential is fully explored.

Controlling random seeds. Manually setting the values of random seeds for random number generators in experiments and varying these values tests the robustness of the results against stochastic variations. It helps to identify if the observed performance is genuinely attributable to the algorithm's efficacy or merely a result of random chance. This practice enhances the credibility of the results.

Replication runs. Stochastic variations can arise from initial model parameter settings, randomness in data shuffling and partitioning, variability in the optimization process such as stochastic gradient descent, and differences in the allocation of computational resources. Experimental replication involves conducting the same experiment multiple times with identical configurations to assess the consistency of the results amidst inherent stochastic variations. This method is crucial in distinguishing between true experimental effects and outcomes influenced by random chance.

Ablation studies. Ablation studies systematically remove parts of the algorithm (e.g., features, layers in a neural network) to understand the contribution of each component to the overall performance. This method helps in identifying the critical elements of the algorithm and can guide further refinement and simplification.

Sensitivity analysis. Sensitivity analysis involves varying the algorithm's parameters or the configuration of its modules to assess the impact on performance. This analysis can reveal dependencies and robustness of the algorithm against changes, providing insights into its behavior under different conditions.

Result averaging and confidence intervals. Averaging results across multiple runs (due to rerunning, different seeds, or cross-validations) and computing mean and standard deviation for selected metrics provides a more stable and reliable estimate of the algorithm's performance. This method helps in summarizing the results and facilitates comparison with other algorithms. Alongside mean performance metrics, reporting confidence intervals provides a statistical range within which the true performance metric is likely to lie. This gives a clearer picture of the variability and reliability of the performance estimates. A narrow confidence interval around the average result suggests high reproducibility and stability of the algorithm's performance, whereas a wide interval signals greater variability and potential sensitivity to experimental conditions.

Statistical testing. Statistical tests can be employed to rigorously evaluate whether the differences in performance metrics between algorithms are statistically significant. This adds a layer of quantitative rigor to the analysis, supporting stronger conclusions about the comparative effectiveness of the algorithms.

3 Experimental Process

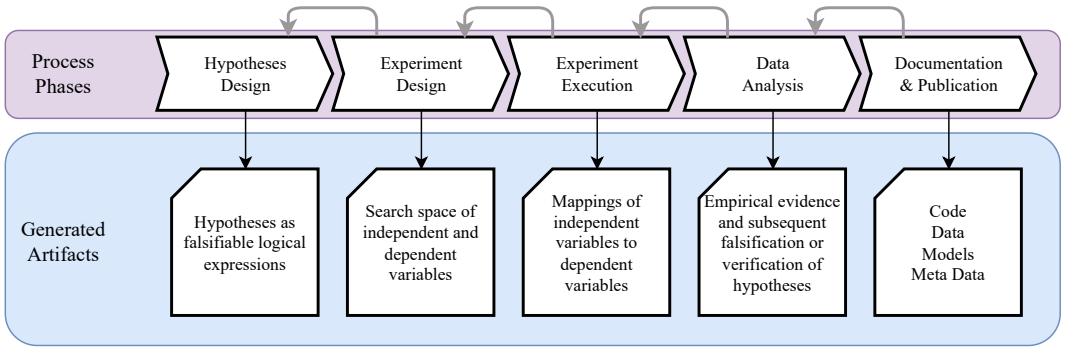


Figure 1 Research process model with process phases and generated artifacts.

We define the experimental process (see Fig. 1) beginning with the formulation of research hypotheses. Based on the hypotheses, experiments are designed. The experimental design defines the necessary input and output of the experiment as well as the metrics to be applied to it to generate empirical evidence. The execution of the experiment generates data, which through an analysis generates evidence. This evidence is directly linked to the hypotheses and used for validation or falsification of them. Results are then documented and published. In the following, we describe the phases in more detail and elaborate our proposed methods to increase empirical research quality. Whilst our research process model shows a linear forward oriented order of process steps, research is an iterative process. Cycles within this process are natural and a step back into a previous phase due to new insight does comply with our process model.

3.1 Formulation of Hypotheses

A hypothesis [1, 14] is a logical statement that postulates a specific relationship among the independent variables X, the control variables C, and the dependent variables Y of an experiment. X and C represent the input variables of an experiment and Y the output observations, see chapter 3.2. This relationship can be expressed in terms of equality, inequality, or other logical constructs that can be evaluated as true or false based on empirical data obtained from the experiment. The hypothesis can be categorized into two main types: null hypothesis H_0 and alternative hypothesis H_a.

The null hypothesis H_0 typically asserts that there is no effect or no difference, or it states a specific value or condition expected in the absence of a relationship among the variables. For example, H_0 could assert that the mean response is not affected by different levels of an independent variable. The use of null hypotheses provides a clear, objective baseline for comparison and ensures the rigor of scientific research by requiring falsifiable claims. It allows for the control of errors and the determination of statistical significance, thereby enhancing the reliability of the research findings.

The alternative hypothesis H_a proposes a relationship or effect that contradicts the null hypothesis. It is usually what the research seeks to support through the data. For instance, H_a might suggest that the mean response for one level of an independent variable is greater than for another level.

Formally, a hypothesis can be written as:

$$H_0 : \phi(X, C, Y) = 0 \tag{1}$$

$$H_a : \phi(X, C, Y) \neq 0 \tag{2}$$

where ϕ is a function that describes the hypothesized relationship or effect among X, C, and Y. The actual form of ϕ depends on the specific nature of the hypothesis being tested. Hypotheses are evaluated by analyzing data collected from experiments to determine whether there is sufficient evidence to reject H_0 in favor of H_a. Here statistical tests should be used as shown in section 3.4.

The hypothesis posits that manipulating or varying the independent variables $\mathbf{X}$ will lead to a change or effect in the dependent variables $\mathbf{Y}$. This relationship is testable, meaning that experimental data can potentially prove the hypothesis true or not true. In accordance with [19] this definition includes that the content of a hypothesis must enable an empirical and isolated evaluation.

In more general terms, a hypothesis must fulfill the following:

Clarity and Specificity: The hypothesis should be clear and precise, making it understandable without ambiguity. It must be specific, detailing the expected relationship between independent and dependent variables.

Testability and Falsifiability: The hypothesis must be testable, meaning that it can be supported or refuted through empirical evidence obtained via experimentation or observation. It must enable the derivation of specific, testable predictions to ensure that the relationship between variables can be empirically evaluated. There must be possible empirical outcomes that could prove the hypothesis wrong.

These criteria ensure that a hypothesis is not only theoretically sound but also practically applicable in guiding experimental design and analysis. In the domain of CPS, research hypotheses commonly focus on comparing a novel method against established baselines or evaluating performance against a predefined threshold for a particular metric on a specific task. These hypotheses can be categorized into qualitative and quantitative types. Qualitative

hypotheses aim to establish a superior/inferior relationship, assessing whether the new method outperforms or underperforms relative to existing approaches without necessarily quantifying the difference. On the other hand, quantitative hypotheses seek to define a measurable relationship, often in terms of specific performance metrics (e.g., accuracy, efficiency, response time). The majority of hypotheses are quantitative. This is because ML experiments typically generate vast amounts of data that can be precisely measured and statistically analyzed, allowing for concrete comparisons and evaluations.

3.2 Experiment Design

The purpose of experiments is to generate empirical evidence about the validity of hypotheses. Hence, experiments need to be designed in a way that they follow the logic of hypotheses and experimental parameters need to be controlled, so that the relation of interest can be investigated in a statistically significant manner.

In ML, an experiment consists of a ML-program (code) which is a concrete implementation of an ML-algorithm, run in a specific environment (hardware and software) and given specific data as input. Following the definition of a hypothesis, let an experiment ϵ be defined as a function that maps a set of independent variables X and a set of control variables C to a set of dependent variables Y.

$$\epsilon : (X, C) \to Y \tag{3}$$

Both X and C can include variables from different domains. If a variable is real-valued, it belongs to the domain $\mathbb{R}$, or a subset thereof, representing continuous values. If a variable is discrete, it is defined over $\mathbb{Z}$, or a specific subset thereof, representing countable values. For categorical variables, we define a set $\mathcal{C}^i = \{Category_1, Category_2, ..., Category_i\}$, where each category represents a distinct qualitative value. The dependent variable(s) Y can also be from one of these domains, depending on the nature of the outcome being measured in the experiment. With $n, m, o, p, q, r \in \mathbb{N}_0$, we define the domains of X, C, and Y as follows:

$$X \subseteq \mathbb{R}^n \cup \mathcal{C}^m \tag{4}$$
$$C \subseteq \mathbb{R}^o \cup \mathcal{C}^p \tag{5}$$
$$Y \subseteq \mathbb{R}^q \cup \mathcal{C}^r \tag{6}$$

The function ϵ maps combinations of independent and control variables to an outcome in Y, modeling the experiment's behavior:

$$\epsilon : ((\mathbb{R}^n \cup \mathcal{C}^m) \times (\mathbb{R}^o \cup \mathcal{C}^p)) \to (\mathbb{R}^q \cup \mathcal{C}^r) \tag{7}$$

To illustrate this, we take a typical ML experiment, where we want to determine the anomaly detection performance of a model. Our control variables C would comprise factors like the model architecture (e.g., convolutional neural network, recurrent neural network), preprocessing techniques (e.g., normalization, dimensionality reduction), and the fixed parameters of the model (e.g., layers, activation functions) that are not subject to change during the experiment. These control variables are kept constant to ensure that the experiment accurately assesses the effect of the independent variables on the performance. The independent variables X could include the hyperparameters of the model that we intend to optimize, such as the learning rate, batch size, or the number of epochs, but also the data which is used or random seed values. These variables are systematically varied to observe their effect on the model's anomaly detection performance. Finally, the dependent variables Y

would represent the outcomes of the experiment, specifically the performance metrics of the anomaly detection model, such as precision, recall, or F1 score. These metrics quantify the model's effectiveness in distinguishing between normal and anomalous instances in the dataset.

In summary, a hypothesis for this ML experiment might propose that certain configurations of the learning rate, batch size, and number of epochs (X) significantly improve the model's anomaly detection performance (Y), under fixed conditions of model architecture, preprocessing techniques, and model parameters (C). This hypothesis can then be tested through a series of experiments, analyzing the collected data to support or refute the proposed relationship.

A commonly used methodology used in this context is the Design of Experiments (DoE) method [5], which offers a systematic approach for the selection of independent and control variables.

3.3 Experiment Execution

The process of conducting experiments involves translating an ML algorithm into an ML program. This program is designed to produce dependent variables from the predetermined independent and control variables. Given that a ML program operates within a complex ecosystem influenced by software and hardware dependencies, it's imperative to structure experiments to ascertain the impact of these dependencies. Additionally, it's crucial to determine the presence of any stochastic behavior in the outcomes, ensuring that the observed results are robust and reliable under varying conditions. Aleatoric and epistemic uncertainty in empirical research [12] must be reduced as much as possible. To mitigate such effects, methods as seen in the state of the art can be applied. Replication runs should be used to ascertain replicable results, whilst multiple random seeds, shuffled data and hyperparameters should be tested to rule out stochastic effects. The use of multiple (standard) data sets enables a broader validity of the research. The decision for these aspects are made during the previous design phase, but can be adapted in an exploratory manner.

We propose to design the experiment as a modular piece of software, with inputs being only the independent and control variables and outputs being the dependent variables to increase transparency and replicability of the results. The data of every experiment shall be saved in a complete and structured manner to enable the following analysis on the relation of independent and dependent variables (see 3.5, 6).

3.4 Statistical Data Analysis

Table 1 Comparison of statistical tests based on distribution and variance.

Distribution	Variances	Appropriate Test
Normal	Equal	Paired t-test
Normal	Unequal	Paired t-test with Welch's correction
Not Normal	Any	Wilcoxon signed-rank test
Mixed	Any	Mann-Whitney U test, Kolmogorov-Smirnov test

Statistical testing [13, 20] is crucial for evaluating research hypotheses, accommodating scenarios both with and without a predefined null hypothesis (H_0). When H_0 is specified, it generally asserts the absence of an effect or relationship, serving as a reference for testing the experimental data. Various statistical tests, such as t-tests, ANOVA, or chi-square tests,

are utilized to compute the probability (p-value) of observing the experimental outcomes if H_0 were accurate. A result is considered statistically significant if the p-value is less than a predetermined threshold (e.g., $\alpha = 0.05$), indicating evidence to reject H_0 in favor of the alternative hypothesis (H_a), which asserts a specific effect or relationship. The choice of the appropriate test depends on the attributes of the data distributions. See table 1 for our suggestions.

In research that does not explicitly use a null hypothesis, the emphasis is on directly evaluating the support for the specific hypothesis under investigation. The statistical analysis aims to quantify the strength and reliability of the observed effects or relationships. Techniques such as confidence intervals, effect sizes, and other statistical measures are applied to assess the hypothesis' credibility. The lack of a null hypothesis underscores the importance of statistical precision in estimating and interpreting the significance as well as the relevance of the study's findings.

Regardless of the presence of a null hypothesis, the core objective of statistical analysis in hypothesis testing is to meticulously quantify the uncertainty surrounding empirical observations. This rigorous approach ensures that conclusions derived from the data are not only based on detected patterns but also statistically substantiated. We provide additional materials, including a code example for designing and statistically testing hypotheses, on GitHub [23].

3.5 Documentation and Publication

During the execution of the experiments data are generated. To enable unambiguous interpretation of the experimental data, all aspects of an experiment need to be documented. We propose that for every run of an experiment all independent, control and dependent variables are documented and that learned models are also saved where possible. This approach guarantees a full traceability of results. Traceable results increase replicability and offer a certain level of quality assurance.

We encourage the application of the FAIR principles [24] for published data to increase the usability of it. In general the documentation should also include the specifications of used software including the software version number and hardware (CPU, GPU, memory). Total needed computation time on the used environment should also be sated.

4 Application to Diagnosis and Planning related Research

Our methodology is versatile and applicable to any research involving ML algorithms. This brief overview of selected studies is designed to give a general idea of the role of ML in diagnosis and planning, demonstrating how our methodology can be applied, rather than serving as an exhaustive explanation.

Traditionally, diagnosis and planning rely on first-principle models and logical expressions about a system. However, these tasks require extensive expert knowledge, and the workload increases significantly as system complexity grows. To address this, ML models have been applied across various tasks, incorporating both symbolic and sub-symbolic techniques into hybrid approaches. For example, ML models can replace initial anomaly detection in system diagnostics, reducing the need for expert intervention. In this context, we highlight selected research that partially employs our recommended methods, demonstrating their applicability within the diagnosis and planning domains.

All the selected research is of empirical nature and based on the formulation of hypotheses, which shall be validated or falsified through empirical evidence. In the field of diagnosis, our works [16, 17, 21] make use of ML algorithms for anomaly detection to be used for subsequent

diagnosis. Here various of the proposed methods such as data shuffling, cross validating, hyperparameter optimization, baseline methods, randomness control and replication runs were used.

In the field of planning, our works [7, 4, 8] use ML algorithms to enhance or replace symbolic approaches for planning tasks related to CPS. Here we used a combination of multiple data sets, baseline methods and hyperparameter optimization to evaluate the performance of the proposed algorithm. Additionally, we performed an ablation study on the observer functions to investigate their impact on the performance in [7].

5 Conclusion and Outlook

In this paper, we delve into the critical aspects of conducting empirical research in the ML domain, focusing on the importance of developing clear hypotheses, designing thorough experiments, executing these with precision, and conducting detailed statistical analysis. Our analysis highlights the absence of standardized methods, leading to significant variations in research quality. The process from formulating hypotheses to analyzing outcomes is fundamental for valid ML research, yet the diversity in research practices signals a need for a more unified approach.

To bridge this gap, we introduce a research process model and a set of guidelines aimed at improving the consistency, reliability, and overall impact of empirical ML research. By adopting these recommendations, the research community can move towards more standardized practices, enhancing the replicability of studies and the reliability of their results.

This paper represents an initial effort to establish unified guidelines for high-quality empirical research within the ML field. We recognize the opportunity to further elaborate on the descriptions of process phases and artifacts, with the goal of crafting a standard process model in future research. Such a model could serve as a foundation for ensuring research quality and facilitating its adaptation to industrial applications. Future research should also include an empirical investigation of the usability and benefits of the proposed process model with regards to replicability and reproducibilty beyond the currently given examples.

6 Checklist

This paper concludes with a comprehensive checklist that encompasses critical aspects of research methodology, including hypothesis formulation, experimental design, execution, data analysis, and documentation. The checklist serves as a systematic guide to ensure thoroughness, clarity, and adherence to scientific standards throughout the research process. It aims to facilitate the validation of the study's methodology and enhance the reliability and impact of its findings. By incorporating these considerations, researchers can bolster the integrity of their work, promoting transparency and reproducibility in the scientific community.

- Falsifiable hypotheses defined
- Independent variables defined
- Control variables defined
- Dependent variables defined
- Baseline models defined
- Multiple data sets selected
- Replication runs (re-runs) performed

- All random seeds set and multiple values tested
- Cross validations over partial data sets performed
- Hyperparameter tuning for every model including baselines performed
- Results averaged (mean and variance) over cross validations, seeds and replications
- Statistical testing of hypotheses performed
- Complete experiment code published
- Software environment published
- Data published (FAIR principles)
- Trained model (weights) published

References

1. Martin W. Bauer. Hypothesis. *The Palgrave Encyclopedia of the Possible*, pages 692–701, 2022. doi:10.1007/978-3-030-90913-0_193.
2. Chris Drummond. Replicability is not reproducibility: Nor is it good science. *Proc. of the Evaluation Methods for MachineLearning Workshop at the 26 th ICML*, 2009. URL: https://www.researchgate.net/publication/228709155_Replicability_Is_Not_Reproducibility_Nor_Is_It_Good_Science.
3. Chris Drummond. Reproducible research: a minority opinion. *Journal of Experimental and Theoretical Artificial Intelligence*, 30:1–11, January 2018. doi:10.1080/0952813X.2017.1413140.
4. Jonas Ehrhardt, René Heesch, and Oliver Niggemann. Learning process steps as dynamical systems for a sub-symbolic approach of process planning in cyber-physical production systems. *Communications in Computer and Information Science*, 1948 CCIS:332–345, 2024. doi:10.1007/978-3-031-50485-3_34/FIGURES/4.
5. Ronald A. Fisher. *The design of experiments*. Oliver & Boyd, 1935.
6. Odd Erik Gundersen and Sigbjørn Kjensmo. State of the art: Reproducibility in artificial intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32:1644–1651, April 2018. doi:10.1609/AAAI.V32I1.11503.
7. René Heesch, Alessandro Cimatti, Jonas Ehrhardt, Alexander Diedrich, and Oliver Niggemann. A lazy approach to neural numerical planning with control parameters. In *ECAI 2024*. IOS Press, 2024.
8. René Heesch, Jonas Ehrhardt, and Oliver Niggemann. Integrating machine learning into an SMT-based planning approach for production planning inc yber-physical production systems. *Communications in Computer and Information Science*, 1948 CCIS:318–331, 2024. doi:0.1007/978-3-031-50485-3_33.
9. Sabine Hoffmann, Fabian Scheipl, and Anne-Laure Boulesteix. Reproduzierbare und replizierbare forschung. *Moderne Verfahren der Angewandten Statistik*, pages 1–28, 2023. doi:10.1007/978-3-662-63496-7_25-1.
10. Keith J. Holyoak and Robert G. Morrison. *The Oxford Handbook of Thinking and Reasoning*. Oxford University Press, November 2012. doi:10.1093/OXFORDHB/9780199734689.001.0001.
11. Matthew Hutson. Artificial intelligence faces reproducibility crisis unpublished code and sensitivity to training conditions make many claims hard to verify. *Science*, 359:725–726, February 2018. doi:10.1126/SCIENCE.359.6377.725/ASSET/65660AD9-910B-4564-8959-7E3192C0286D/ASSETS/GRAPHIC/359_725_F2.JPEG.
12. Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110:457–506, March 2021. doi:10.1007/S10994-021-05946-3/FIGURES/17.
13. Tatjana Ille and Natasa Milic. Statistical tests. *Encyclopedia of Public Health*, pages 1341–1344, 2008. doi:10.1007/978-1-4020-5614-7_3349.
14. Hiroshi Ishikawa. Hypothesis. *Studies in Big Data*, 139:33–69, 2024. doi:10.1007/978-3-031-43540-9_2/FIGURES/16.

15 Herbert Keuth. Karl popper: Logik der forschung. *Karl Popper: Logik der Forschung*, 2007. doi:10.1524/9783050050188/HTML.

16 Lukas Moddemann, Henrik Sebastian Steude, Alexander Diedrich, and Oliver Niggemann. Discret2di - deep learning based discretization for model-based diagnosis. *IFAC-PapersOnLine*, 58:640–645, January 2024. doi:10.1016/J.IFACOL.2024.07.291.

17 Lukas Moddemann, Henrik Sebastian Steude, Alexander Diedrich, Ingo Pill, and Oliver Niggemann. Extracting knowledge using machine learning for anomaly detection and root-cause diagnosis | request pdf. In *IEEE ETFA 2024 - IEEE International Conference on Emerging Technologies and Factory Automation*, 2024. URL: https://www.researchgate.net/publication/381574722_Extracting_Knowledge_using_Machine_Learning_for_Anomaly_Detection_and_Root-Cause_Diagnosis.

18 Committee on Reproducibility and Replicability in Science. *Understanding Reproducibility and Replicability - Reproducibility and Replicability in Science - NCBI Bookshelf*. National Academies Press (US), 2019. URL: https://www.ncbi.nlm.nih.gov/books/NBK547546/.

19 Friedrich Rapp. The methodological symmetry between verification and falsification on jstor. *Journal for General Philosophy of Science*, 6:139–144, 1975. URL: https://www.jstor.org/stable/25170349?seq=1.

20 Hartmut Schiefer and Felix Schiefer. Statistical tests. *Statistics for Engineers*, pages 69–93, 2021. doi:10.1007/978-3-658-32397-4_5.

21 Henrik Sebastian Steude, Lukas Moddemann, Alexander Diedrich, Jonas Ehrhardt, and Oliver Niggemann. Diagnosis driven anomaly detection for cyber-physical systems. *IFAC-PapersOnLine*, 58:13–18, January 2024. doi:10.1016/J.IFACOL.2024.07.186.

22 Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus Robert Müller. Towards crisp-ml(q): A machine learning process model with quality assurance methodology. *Machine Learning and Knowledge Extraction*, 3:392–413, March 2020. doi:10.3390/make3020020.

23 Daniel Vranjes. Empirical machine learning research guide, 2024. URL: https://github.com/danvran/EmpiricalMachineLearningResearchGuide.

24 Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C. t Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan Van Der Lei, Erik Van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The fair guiding principles for scientific data management and stewardship. *Scientific Data 2016 3:1*, 3:1–9, March 2016. doi:10.1038/sdata.2016.18.

Diagnosing Multi-Agent STRIPS Plans

Avraham Natan
Ben-Gurion University of the Negev, Beersheba, Israel

Roni Stern
Ben-Gurion University of the Negev, Beersheba, Israel

Meir Kalech
Ben-Gurion University of the Negev, Beersheba, Israel

William Yeoh
Washington University in St. Louis, MO, USA

Tran Cao Son
New Mexico State University, Las Cruces, NM, USA

Abstract

The increasing use of multi-agent systems demands that many challenges be addressed. One such challenge is diagnosing failed multi-agent plan executions, sometimes in system setups where the different agents are not willing to disclose their private actions. One formalism for generating multi-agent plans is the well-known MA-STRIPS formalism. While there have been approaches for delivering as robust plans as possible, we focus on the plan execution stage. Specifically, we address the problem of diagnosing plans that failed their execution. We propose a Model-Based Diagnosis approach to solve this problem. Given an MA-STRIPS problem, a plan that solves it, and an observation that indicates execution failure, we define the MA-STRIPS diagnosis problem. We compile that problem into a boolean satisfiability problem (SAT) and then use an off-the-shelf SAT solver to obtain candidate diagnoses. We further expand this approach to address privacy by proposing a distributed algorithm that can find these same diagnoses in a decentralized manner. Additionally, we propose an enhancement to the distributed algorithm that uses information generated during the diagnosis process to provide significant speedups. We found that the improved algorithm runs more than 10 times faster than the basic decentralized version and, in one case, runs faster than the centralized algorithm.

2012 ACM Subject Classification Hardware → Bug detection, localization and diagnosis; Computing methodologies → Multi-agent systems; Computing methodologies → Distributed algorithms; Security and privacy → Domain-specific security and privacy architectures

Keywords and phrases Model-based diagnosis, Multi-agent systems, Distributed diagnosis, Privacy

Digital Object Identifier 10.4230/OASIcs.DX.2024.8

1 Introduction

In recent years, Multi-Agent Systems (MAS) have seen a significant increase in applications in different areas such as logistics, transportation, public services, and more. In many MAS applications, the agents are tasked with meeting a set of predefined goals. To achieve this, each agent is given a series of actions to perform. This is called a Multi-Agent Plan (MAP). Figure 1 demonstrates a simple example where the goal is to move a package from a location in city 1 to a location in city 2, where trucks can travel between locations in the cities and airplanes can fly between the cities.

Multi-Agent STRIPS (MA-STRIPS) is a well-established formalism for MAP [3, 4, 2]. Most efforts in the MA-STRIPS formalism focused on various planning algorithms and heuristics. These advances have contributed to generating efficient plans.

When such a plan is executed, however, faults may occur unexpectedly. This may lead to unmet goals. Some approaches address this challenge during the planning phase by


© Avraham Natan, Roni Stern, Meir Kalech, William Yeoh, and Tran Cao Son;
licensed under Creative Commons License CC-BY 4.0



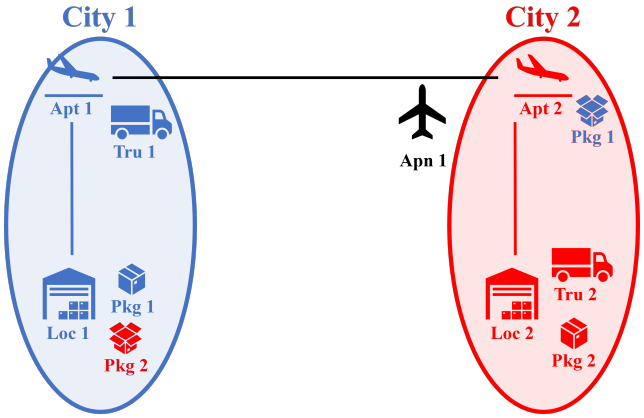


Figure 1 An illustration of a package delivery problem. Every truck can move between the locations of its respective city, and the airplane can move only between airports. The goal is to move the packages between initial locations (closed packages) and goal locations (open packages).

computing robust plans [19]. Execution faults, however, may still occur. In such cases, it is important to understand which actions or agents failed. This is the task of *Fault Diagnosis* [22]. Fault Diagnosis explains the inconsistencies between the expected outcome of a system run and the observed outcome. It uses knowledge about the problem, the plan, and the execution.

Our **first contribution** is to define the problem of diagnosing MA-STRIPS plan executions. Our **second contribution** is a centralized MBD-based algorithm called MAS-DX for solving this problem. Our **third contribution** addresses privacy concerns, allowing each agent to keep private information. The algorithm, called Dec-MAS-DX, is a decentralized MBD-based algorithm that works in three phases. First, the problem is divided into local problems, where each local problem is relevant to a specific agent. Then, problems are solved to generate local diagnoses. Finally, the local diagnoses are combined into the final set of diagnoses. We propose an enhancement to this algorithm that intelligently coordinates the diagnosis process. As the **fourth and final contribution**, we provide empirical evaluation on eight MA-STRIPS domains. We conclude that the centralized algorithm (MAS-DX) has better runtime overall, while the distributed algorithm (Dec-MAS-DX) can address privacy and even outperforms in terms of run-time for one of the domains.

2 Literature Review

2.1 Related Work

Within the MA-STRIPS formalism [3, 4, 27], advances have tackled different problems. Some examples include distributed [20, 2] and heuristic-based [26, 28] planning. A recent work addressed learning the models of the actions [17].

In this work, we assume the plan is given and focus on the diagnosis part. Diagnosis of Multi-Agent Systems has been studied in various settings w.r.t. different assumptions about the agents and how they can fail. Some studies aim to isolate which actions have failed during execution [8, 23, 7, 16, 29, 18]. Others aim to isolate which agents' beliefs made them choose actions that led to coordination failures [6, 11, 12]. We refer to [13] for a comprehensive survey of previous work in this field.

A very recent work has considered diagnosing plans within MA-STRIPS formalism [25]. However, they focused on extending the MA-STRIPS language to support common-sense fluents and how these can be used for diagnosis purposes. We, on the other hand, focus solely on the diagnosis process.

2.2 Background

MA-STRIPS is a formalism for defining multi-agent planning problems in a classical planning setting, i.e., where actions are deterministic, and states are represented by sets of discrete facts.

A planning problem in MA-STRIPS is defined by a tuple $\Pi = \langle F, k, \{A_i\}_{i=1}^{k}, I, G\rangle$ where F is a set of fluents, k is the number of agents we are planning for, A_i is the set of actions agent i can perform, I is the initial state, and G is the desired goal. A fluent $f \in F$ represents a fact that may or may not be true in a given state. The initial state I is the set of all the true fluents in the initial state, and the goal G is the set of fluents we want to achieve. Every action $a \in A_i$ is defined by a tuple $\langle name(a), pre(a), \mathit{eff}(a)\rangle$, where $name(a)$ is the name of the action, $pre(a)$ is the action's preconditions, and $\mathit{eff}(a)$ is the action's effects. The preconditions and effects of an action a are sets of *literals*, where a literal is either a fluent or its negation. An action a is *applicable* in a state s if:

- for every literal $l \in pre(a)$ such that $l = f$, the fluent f is in s.
- for every literal $l \in pre(a)$ such that $l = \neg f$, the fluent f is not in s.

Applying a to a state s results in a state denoted $a(s)$, where:

- s includes fluents f such that $l = f$ is in $\mathit{eff}(a)$.
- s does not include fluents f such that $l = \neg f$ is in $\mathit{eff}(a)$.

In a multi-agent setting, it is common for agents to be able to act concurrently. The concurrent execution of a set of actions is represented by a *joint action*, which is an k-ary vector $\mathbf{a} \in \{A_1 \cup \{nop\}\} \times \{A_2 \cup \{nop\}\} \times \cdots \times \{A_k \cup \{nop\}\}$, where *nop* represents a no-op, i.e., that the corresponding agent does not perform any action. The preconditions and effects of a joint action can generally be different from the preconditions and effects of its constituent single-agent actions [1, 24]. For simplicity, we will assume in this work that the preconditions and effects of a joint action are the corresponding union of the joint action's constituent single-agent actions. This assumption limits our approach to MA-STRIPS domains, where actions do not require more than one agent to be performed. However, the set of MA-STRIPS domains is still large enough for our approach to be relevant despite this limitation. To avoid ambiguity, we also require that executed joint actions are *well-defined* [5], which means that their effects are consistent.

▶ **Definition 1 (Multi-Agent Plan).** *For an MA-STRIPS problem* $\Pi = \langle F, k, \{A_i\}_{i=1}^{k}, I, G\rangle$, *A plan* π *is a sequence of* n *joint actions* $\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n$ *such that (1)* $\boldsymbol{a}_1$ *is applicable in the initial state* I*, (2) for each* $1 < t \leq n$, $\boldsymbol{a}_t$ *is applicable in the state* $\boldsymbol{a}_{t-1}(...\boldsymbol{a}_1(I)...)$*, and (3)* $G \subseteq \boldsymbol{a}_n(\boldsymbol{a}_{n-1}(...\boldsymbol{a}_1(I)...))$.

Table 1 shows an example of a plan that solves the MA-STRIPS problem presented in Figure 1. The trucks and the airplane collaborate in order to deliver the packages to their destinations. Every row in the table outlines a joint action that consists of all the actions in that row. They can be performed together since we assume the effects of these actions correspond to the union of the joint action's constituent single-agent actions. A counter-example for that is action four of the $apn1$ and action three of $tru2$. One of the effects of the action $unload(tru2, p2)$ is that $p2$ can be picked by other vehicles in the future. Only then, $apn1$ can execute its action $load(apn1, p2)$. Hence, those two actions are physically impossible to perform simultaneously and cannot be parts of the same joint action.

Table 1 Example of an MA-STRIPS plan.

#	Airplane 1	Truck 1	Truck 2
1	nop	drive(tru1,loc1)	load(tru2,p2)
2	nop	load(tru1,p1)	drive(tru2,apt2)
3	nop	drive(tru1,apt1)	unload(tru2,p2)
4	load(apn1,p2)	unload(tru1,p1)	nop
5	fly(apn1,apt1)	nop	nop
6	unload(apn1,p2)	nop	nop
7	load(apn1,p1)	load(tru1,p2)	nop
8	fly(apn1,apt2)	drive(tru1,loc1)	nop
9	unload(apn1,p1)	unload(tru1,p2)	nop

3 Problem Definition

Executing a plan π of n joint actions on an initial state I incurs a series of $n+1$ states, where each state essentially differs from the previous one by the effects of the corresponding joint action. We call such a series a *Trajectory*.

▶ **Definition 2 (Trajectory).** *Given an initial state I and a plan $\pi = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, a* trajectory *$T(I, \pi)$ is an alternating sequence of states and joint actions $(I = s_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n, s_n)$, where s_{i-1} and s_i are the states before and after joint action $\boldsymbol{a}_i$, respectively. The triplet $(s_{i-1}, \boldsymbol{a}_i, s_i)$ is referred as the i^{th} transition of the trajectory. In the rest of the paper, unless required for context, we will denote $T(I, \pi)$ simply as T.*

Since MA-STRIPS domains are deterministic, the initial state I and plan π induce a unique trajectory where for each state s_i it holds that $s_i = \mathbf{a}_i(...\mathbf{a}_1(I)...)$. We call this trajectory the *expected trajectory* and denote it as $T_{exp}(I, \pi)$. This trajectory, however, may not represent the actual execution of the plan. We focus on the cases where, at some point, the trajectory differs from $T_{exp}(I, \pi)$. In this work, we assume there are two reasons for this difference. The first reason is an action that was applicable in the current state but was not applied or did not have its intended effects. We refer to such an action as a *faulty action*. The second reason is an action that is not applicable in the current state. We refer to it as a *conflicted action*. Let us formally define them.

▶ **Definition 3 (Faulty Action, Conflicted Action).** *Given a plan π for an MA-STRIPS problem $\Pi = \langle F, k, \{A_i\}_{i=1}^{k}, I, G\rangle$, a trajectory $T(I, \pi)$ and a transition $(s, \boldsymbol{a}, s') \in T(I, \pi)$, a single-agent action $a \in \boldsymbol{a}$ is said to be faulty if it is applicable in state s, but $a(s) \neq s'$, and is said to be conflicted if it is not applicable in state s. We denote the set of faulty and conflicted actions in trajectory T as $F(T)$ and $C(T)$, respectively.*

In this work, we assume that the root causes of conflicted actions are faulty actions caused by an internal error in one of the agents and not by causes related to the environment. Consequently, in every trajectory $T(I, \pi) \neq T_{exp}(I, \pi)$, we assume there must be at least one faulty action.

Multi-agent systems commonly have monitoring mechanisms that collect *observations* during plan execution in order to monitor execution and detect discrepancies between expected and executed trajectories that may lead to system failures. In our work we define observation as states that had their fluents observed.

▶ **Definition 4 (Observation).** *An observation O is an alternating sequence of the form $(o_0, \boldsymbol{a_1}, o_1, \ldots, \boldsymbol{a_n}, o_n)$ where o_i is a set of literals and $\boldsymbol{a_i}$ is a joint action. An observation is* consistent *with a trajectory $T = (s_0, \ldots, s_n)$ if they consist of the same sequence of joint actions and for every i either o_i is null (i.e., not observed) or $o_i = s_i$. Otherwise, we say that the observation is inconsistent with T.*

In this work, we assume that each state is fully observed or not observed. We leave cases where some of the states' fluents are observed as future work.

▶ **Example 5.** For example, consider the third action of $tru2$ in Table 1 ($unload(tru2, p2)$). Observing $p2$ at $apt2$ after the action execution is consistent with the plan since one of the action's effects is that $p2$ is at $apt2$. An example of an inconsistent observation is to observe that $p2$ is inside $tru2$. In that case, the explanation is that $tru2$ could not carry its action to unload $p2$.

In the context of diagnosis, there are 2 main approaches for models used to describe the behaviour of faulty actions: *weak fault model* (WFM) and *strong fault model* (SFM). Weak fault model does not specify the behaviour of a faulty action and is essentially the negation of the normal behaviour of an action. Although simpler to implement, WFM usually means longer run-time for computing a diagnosis due to larger size of different configurations of faulty actions. Strong fault model, on the other hand, specifies exactly how a faulty action should behave, and by that imposing restrictions on the states of other actions, thus contributing to a smaller space of possible faulty action configurations. However, SFM usually requires additional assumptions about the system, and in cases where there is more than one candidate fault model, this can actually contribute to much higher complexity of the diagnosis process.

In this work, we define strong model both for the behaviour of faulty actions and for conflicted actions, and denote them by M_f and M_c, respectively. In the context of MA-STRIPS, given an action a, $M_f(a)$ and $M_c(a)$ are sets of literals in the same way that $\mathit{eff}(a)$ is. Applying a faulty action a on s is denoted as $(M_f(a))(s)$ and the resulting state is different than the state $a(s)$. The definition of fault and conflict models allows our approach to be generalized to different MA-STRIPS domains, since then all one has to do is define the specifics of M_f and M_c. The fault and conflict models we will explore in this work are *noeffect* models. It means that if an action is faulty or conflicted, its effect will not occur. In other words, for such models, $M_f = M_c = \emptyset$. Effects that do not occur are regarded as different from those that should have occurred according to the MA-STRIPS specification.

When a fault in the execution of a plan π from starting state I occurs, the observation O, collected during the execution, is inconsistent with the expected trajectory $T_{exp}(I, \pi)$. This means that for at least one transition $(s_i, \mathbf{a}, s_{i+1})$ in $T_{exp}(I, \pi)$, it holds that $s_{i+1} \neq o_{i+1}$. This inconsistency indicates that at least one action was faulty when executing π. In that case, the observation O represents a trajectory T' that is different from $T_{exp}(I, \pi)$. In fact, whenever a set of actions becomes faulty, this corresponds to a trajectory that is different from the expected one. Let us define the set of possible trajectories $\mathcal{T}(I, \pi)$:

▶ **Definition 6 (The set of possible trajectories).** *For a starting state I and a plan π, the set of possible trajectories $\mathcal{T}(I, \pi)$ is the set of every possible trajectory that can result by an execution of the plan π where some (or none) of the actions are faulty.*

Note two things. First, $T_{exp}(I, \pi) \in \mathcal{T}(I, \pi)$ by definition. Second, since this definition is not in the context of fault and conflict models, it includes trajectories that, given a specific fault and conflict models, might be impossible. Therefore, we further define the set of trajectories for fault and conflict models M_f and M_c:

▶ **Definition 7 (The set of trajectories for M_f and M_c).** *For a starting state I, a plan π, and fault and conflict models M_f and M_c, the set of trajectories $\mathcal{T}(I, \pi, M_f, M_c)$ is the set of every trajectory $\mathcal{T}(I, \pi)$ where for every transition $(s_i, \boldsymbol{a}, s_{i+1})$, either $s_{i+1} = o_{i+1}$, $(M_f(a))(s_i) = o_{i+1}$ or $(M_c(a))(s_i) = o_{i+1}$.*

We can now define the MA-STRIPS diagnosis problem. An MA-STRIPS diagnosis problem arises when the observation O is not consistent with the expected trajectory $T_{exp}(I, \pi)$. A diagnosis is then defined to be a set of actions that, if assumed to be faulty or conflicted, make the O consistent with $T_{exp}(I, \pi)$. Formally:

▶ **Definition 8 (MA-STRIPS Diagnosis problem).** *An MA-STRIPS diagnosis problem is defined by a tuple $\langle \Pi, \pi, O, M_f, M_c \rangle$, where $\Pi = \langle F, k, \{A_i\}_{i=1}^{k}, I, G \rangle$ is an MA-STRIPS problem, π is a plan that solves it, O is the observation received by executing π from I, M_f is the fault model and M_c is the conflict model. A diagnosis problem arises when O is not consistent with $T_{exp}(I, \pi)$, or in other words, there exists a transition $(s_i, \boldsymbol{a}, s_{i+1})$ such that $s_{i+1} \neq o_{i+1}$. A diagnosis for an MA-STRIPS diagnosis problem is a set of single agent actions $\omega = \{a_{i_1}, a_{i_2}, ...\}$ from the plan π such that there exists a trajectory $T \in \mathcal{T}(I, \pi, M_c, M_f)$ that is consistent with O where $F(T) = \omega$.*

We continue to describe our approach for solving this problem, which involves modelling the problem as a SAT problem.

4 Centralized diagnosis approach

In this section, we present MAS-DX, a centralized approach for solving a MA-STRIPS diagnosis problems. MAS-DX formulates the MA-STRIPS Diagnosis problem as a classical Model-Based Diagnosis (MBD) problem and then solves it using an off-the-shelf MBD algorithm [22]. First, we provide a brief background on MBD.

4.1 MBD Background

An MBD problem is a tuple $\langle SD, COMPS, OBS \rangle$ where SD, $COMPS$, and OBS denote the system description, the set of components, and the observations, respectively. In a Weak Fault Model (WFM) setting, SD takes into account that some components might be faulty without specifying a fault mode. In that case, the unary predicate $h(\cdot)$ on the components specifies that $h(c)$ is true if component c is healthy, and $\neg h(c)$ is true if c is not healthy. When the assumption that all components are healthy is inconsistent with the system model and observed system behavior, a diagnosis problem arises. This is formalized as follows:

$$SD \wedge \bigwedge_{c \in COMPS} h(c) \wedge OBS \vdash \bot$$

where $\vdash$ means to entail and $\bot$ means dissatisfaction. By that, $\vdash \bot$ means that the formula cannot be satisfied, or in other words, inconsistent.

Diagnosis approaches try to find diagnoses, which are possible ways to explain the above inconsistency by assuming that some components are not healthy, and that adhere to the principle of parsimony. Such approaches define diagnosis as a set of components Δ if:

$$SD \wedge \bigwedge_{c \in \Delta} \neg h(c) \wedge \bigwedge_{c \notin \Delta} h(c) \wedge OBS \nvdash \bot$$

and where for each $\Delta' \subseteq \Delta$,

$$SD \wedge \bigwedge_{c \in \Delta'} \neg h(c) \wedge \bigwedge_{c \notin \Delta'} h(c) \wedge OBS \vdash \bot$$

This is also known as minimal subset diagnosis.

Strong fault model, on the other hand, assumes a model f that defines the effects of a faulty action. In that case, a diagnosis δ will be a set of components such that:

$$SD \wedge \bigwedge_{c \in \Delta} f(c) \wedge \bigwedge_{c \notin \Delta} h(c) \wedge OBS \nvdash \bot$$

In this work we assume strong fault models, as discussed in section 3. What this essentially means is that SD extends the healthy behaviour specification with specifications of faulty behaviour [9].

4.2 MA-STRIPS Diagnosis as MBD

MAS-DX formulates a given MA-STRIPS Diagnosis problem as an MBD problem by specifying SD, COMPS, and OBS as follows. OBS are the sets of literals $O = (o_0, \boldsymbol{a_1}, o_1, \ldots, \boldsymbol{a_n}, o_n)$ observed in the plan execution observation as defined in Definition 4. The system components are mapped to the single-agent actions in the plan: $COMPS = \{\{a_{i,t}\}_{i=0}^{k}\}_{t=0}^{n}$, where k denotes the number of agents and n the length of the plan. Three health-state predicates $h(\cdot)$, $f(\cdot)$ and $c(\cdot)$ over the action space specify whether an action is healthy, faulty, or conflicted. The system description SD specifies how the fluents of a state in time t change given a single-agent action a_t. Fluents that are not in the effects of any single-agent action at time t retain their current values. Fluents that are in the effects of an action change their values according to the health state of the action. If the action is normal, the fluent changes its value according to the effects $\mathit{eff}(a_t)$. If the action is faulty or conflicted, the fluent changes its value according to the fault or conflict model, respectively.

To solve the resulting MBD problem, MAS-DX compiles it into a Boolean satisfiability problem (SAT). A SAT problem is defined by a conjunction of boolean literals, and a solution is an assignment of true or false values to those literals, such that the entire formula is true [10]. In the next section, we show how MAS-DX formulates the different parts of our MBD problem as SAT Boolean formulas.

4.3 MA-STRIPS as SAT

4.3.1 Fluents as SAT variables

Recall that in our MA-STRIPS domain, every state is defined as a set of fluents. Thus, for every time-step t, we define for each fluent $f \in F$ a Boolean variable f_t. This allows us to model every state s as a conjunction of Boolean variables $s_t = \bigwedge_{1 \leq j \leq |F|} f_t^j$. Setting f_t^j to true means that the fluent f^j was part of s_t.

4.3.2 COMPS as SAT variables

Since we want to find the faulty actions, we model COMPS as the set of actions $COMPS = \{\{a_{i,t}\}_{i=0}^{k}\}_{t=0}^{n}$, where k denotes the number of agents and n the length of the plan.

4.3.3 OBS as SAT assignments

We encode every observed fluent by setting its corresponding Boolean variable value to true or false. Setting f_t^j to true represents that fluent f^j was observed as part of state s_t.

4.3.4 SD as a conjunction of SAT formulas

Defining SD essentially means defining how the values of the temporal variables f_t change in every time step t. This is determined by the health state of an action $a_{i,t}$. To that end, we model SD as a conjunction of Boolean formulas.

For each action $a_{i,t}$, we define how each health state entails the values of fluents in the preconditions of the action. Formally:

1. $h(a_{i,t}) \rightarrow \bigwedge_{f^j \in pre(a_{i,t})} f_t^j$
2. $f(a_{i,t}) \rightarrow \bigwedge_{f^j \in pre(a_{i,t})} f_t^j$
3. $c(a_{i,t}) \rightarrow \neg(\bigwedge_{f^j \in pre(a_{i,t})} f_t^j)$

Note that normal and faulty actions entail the same fluent values. This is because what differentiates the two health states is whether the effects occur according to the plan or the fault model.

For each action, the health predicates are mutually exclusive, meaning that exactly one of them must be true:

4. $h(a_{i,t}) \otimes f(a_{i,t}) \otimes c(a_{i,t})$

The second group of formulas we define are the transition formulas. For each variable f_t^j, we specify how its value changes in the next state s_{t+1}. We distinguish between four types of variables in time step t:

- variables not in any action effects at time t.
- variables in action effects of a normal action at time t.
- variables in action effects of a faulty action at time t.
- variables in action effects of a conflict action at time t.

For each variable type, we define the transition formulas. We begin by defining the transition of variables that are not in any action effects. We do not assume any exogenous events in our work and therefore, in that case the values stay the same as the values of the previous state. Formally:

5. $\bigwedge_{f^j \notin \bigcup_{i=1}^{k} \mathit{eff}(a_{i,t})} f_{t+1}^j \leftrightarrow f_t^j$

Values of variables that are in the effects of normal actions are set in the next state to the values according to the actions' effects. Formally:

6. $h(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathit{eff}(a_{i,t})} f_{t+1}^j$

Values of variables included in faulty and conflicted actions are set in the next state to the values according to the respective fault and conflict models. Formally:

7. $f(a_{i,t}) \rightarrow \Phi_f(i,t)$

8. $c(a_{i,t}) \rightarrow \Phi_c(i,t)$

where $\Phi_f(i,t)$ and $\Phi_c(i,t)$ represent the formulas encoding the fault and conflict models M_f and M_c.

The fault and conflict models are general and can be defined independently from the model of the system description. In this work, we explore the *noeffect* fault and conflict models, defined as such:

9. $\Phi_f(i,t) \rightarrow \bigwedge_{f^j \in \mathit{eff}(a_{i,t})} f_{t+1} \leftrightarrow f_t$
10. $\Phi_c(i,t) \rightarrow \bigwedge_{f^j \in \mathit{eff}(a_{i,t})} f_{t+1} \leftrightarrow f_t$

Using those formulas, we define the complete SD model:

$$
\begin{aligned}
SD = \bigwedge_{t=1}^{n} \Big(\Big(& \bigwedge_{f^j \notin \bigcup_{i=1}^{k} \mathit{eff}(a_{i,t})} f^j_{t+1} \leftrightarrow f^j_t \Big) \\
& \wedge \Big(h(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathit{eff}(a_{i,t})} f^j_{t+1} \Big) \\
& \wedge \Big(f(a_{i,t}) \rightarrow \Phi_f(i,t) \Big) \\
& \wedge \Big(c(a_{i,t}) \rightarrow \Phi_c(i,t) \Big) \\
& \wedge \Big(h(a_{i,t}) \otimes f(a_{i,t}) \otimes c(a_{i,t}) \Big) \Big)
\end{aligned}
$$

MAS-DX solves the resulting SAT problem using an off-the-shelf SAT solver, and yields a set of boolean assignments that satisfy the SD model. Each one of these assignments can be translated into a diagnosis. Given an assignment ψ and a variable $a_{i,t}$, action a_i that was executed in time t is healthy if $h(a_{i,t}) = \mathit{true}$, faulty if $f(a_{i,t}) = \mathit{true}$, and conflicted if $c(a_{i,t}) = \mathit{true}$. Actions with a faulty health state are the diagnosis of the problem.

▶ **Example 9.** As an example, consider Table 1. Let us assume that action 2 of agent 2 ($a_{2,2} = \mathit{drive}(\mathit{tru}2, \mathit{apt}2)$) failed, causing action $a_{2,3}$ to become conflicted. Because of that, agent 0 (the airplane) will not execute actions $a_{0,4}, a_{0,6}$, since the package $p2$ will not be at the airport. Consequently, agent 1 (truck 1) will fail its actions $a_{1,7}, a_{1,9}$, and the package $p2$ will be observed inside agent 2 at the time of the task execution. In this scenario, MAS-DX will assign a faulty health state to action $a_{2,2}$, since the preconditions will be valid, but the effects will not be as expected. This will be done using formulas $2, 47$ in the problem model. MAS-DX will continue assigning health state values in the same manner, returning a solution where action $a_{2,2}$ is faulty and actions $a_{2,3}, a_{0,4}, a_{0,6}, a_{1,7}, a_{1,9}$ are conflicted.

4.3.5 Proofs of soundness and completeness

▶ **Theorem 10** (Soundness). *Every variable assignment returned by MAS-DX corresponds to a valid diagnosis.*

Proof. Let ψ be an assignment of health state values returned by MAS-DX. To prove the theorem, we show that the values of the health states of the actions are consistent with the observations. In ψ, the values of the observations are already set at the time of the problem modelling. Meaning that $\{f^1_0, \ldots, f^{|F|}_0, \ldots, f^1_t, \ldots, f^{|F|}_t, \ldots, f^1_{t'}, \ldots, f^{|F|}_{t'}, \ldots, f^1_n, \ldots, f^{|F|}_n\}$ which are the values of the fluents in the observed states are set (which means some states that are not observed do not have set values).

Following Formula 4, for every time step and for every agent, exactly one of the health state variables $h(a_{i,t}), f(a_{i,t}), c(a_{i,t})$ is true, where $a_{i,t}$ is the action that agent i was planned to execute at time step t. In addition, Formulas $1-3, 6-8$ mean that the fluent variables of the preconditions and the effects are consistent with the chosen health states. These fluents are fluents with indices t and $t+1$ that correspond to states S_t and S_{t+1}. In turn, those

equations mean that each value of these fluents is consistent with the actions in time steps $t-1$ and $t+1$. In addition, fluents that are not part of effects or preconditions in time step t, get in time step $t+1$ the same values. This is ensured by Formula 5.

Following the same logic, the values reach consistency with the fluent values that were set as part of the observation, during the modelling of the problem. This means that the health state values are consistent with the observations, which is the definition of a diagnosis. ◀

▶ **Theorem 11** (Completeness). *MAS-DX returns a variable assignment for every valid diagnosis.*

Proof. Suppose a valid diagnosis d for which the assigned action health state values are consistent with the observations, and for which, the observations consist of the following observed states: $\{S_0, \ldots, S_t, \ldots, S_{t'}, \ldots, S_n\}$. MAS-DX encodes them as fluent variables with set values $\{f_0^1, \ldots, f_0^{|F|}, \ldots, f_t^1, \ldots, f_t^{|F|}, \ldots, f_{t'}^1, \ldots, f_{t'}^{|F|}, \ldots, f_n^1, \ldots, f_n^{|F|}\}$.

Suppose that according the d, the first faulty action is action $a_{i,t}$. Throughout the assignment process, MAS-DX assigns variables to actions and state fluents such that all actions before $a_{i,t}$ are healthy and all fluents up to $f_t^1, \ldots, f_t^{|F|}$ change according to formulas $1, 5, 6$. At time t, MAS-DX considers the fluents $f_t^1, \ldots, f_t^{|F|}$ that represent state S_t. Their values, are consistent with the formulas defining the preconditions of action $a_{i,t}$. Thus, this action can not be conflicted, and the other two options are faulty or healthy. Each option influences the values of the fluents $f_{t+1}^1, \ldots, f_{t+1}^{|F|}$ that are part of the effects of $a_{i,t}$. From this point on, MAS-DX continues to set different values to state variables, representing the states $S_{t+1}, \ldots S_n$. Following the same process, MAS-DX will check its value assignment with all of the fluents that represent the observed states, and since d is a valid diagnosis, one of these checks will satisfy all of the value assignments. ◀

5 Distributed Diagnosis Approach

MAS-DX is designed to be executed by a central diagnoser that has full access to the information defining the diagnosis problem. The leading assumption here is that the system has all the relevant information in one place, or, that once a diagnosis process begins, the agents voluntarily submit any private information that is required. Thanks to that assumption, MAS-DX is able to model the entire problem as MBD and solve it.

Relaxing such an assumption allows addressing problems where the system description (e.g., the actions of the agents and the fluents) is not available to a central diagnoser. However, the approach we propose in section 4 is not suitable for such cases. Consider, for example, Figure 1. There could be a case where the trucks do not share information about their private actions with an outside diagnoser. We could try to use a diagnoser per truck, but then the private information of other trucks will not be available, which will lead to incorrect modelling of each one of the per-truck diagnosers.

To address this, we propose Dec-MAS-DX, a distributed diagnosis approach for solving MA-STRIPS diagnosis problems. In Dec-MAS-DX, each agent is associated a diagnoser, that models the information available to the agent (private and public). Dec-MAS-DX consists of two main parts. The first part, where each agent attempts to infer diagnoses based on the public information combined with its private information. That information includes the fluents and actions relevant to that agent. At this point, the computed diagnoses are not yet correct - there are duplicates and some diagnoses contradict each other. In the second part, the agents combine these diagnoses in order to converge to diagnoses that are not contradicting each other, and also do not have duplicates.

5.1 Local MA-STRIPS Diagnosis

Prior work on MA-STRIPS used the notion of private and public fluents and actions for each agent [14, 15]. A fluent is private for agent i if it does not appear in the preconditions or effects of any other agent's action. An action is private for agent i if its preconditions and effects only include fluents that are private for that agent. We draw from this definition to define in a similar manner the notions of *relevant fluents* and *relevant actions*.

Unlike private fluents and actions, relevant fluents and actions are defined with respect to the specific plan the agents attempted to execute. This distinction is important for two reasons. First, it means that the plan π, generated as the solution to an MA-STRIPS problem Π, has a major role in defining how the SAT modelling will look like. Second, it means that different plans will define different sets of actions and fluent as relevant to different agents.

▶ **Definition 12 (Relevant Fluent).** *A fluent $f \in F$ is defined as relevant to agent i if $\exists a \in A_i : f \in pre(a) \cup \mathit{eff}(a) \vee \neg f \in pre(a) \cup \mathit{eff}(a)$. We denote the set of relevant fluents to agent i as $\mathcal{R}_i(F)$.*

▶ **Definition 13 (Relevant Action).** *An action a is defined as relevant to agent i if $\exists f \in \mathcal{R}_i(F) : f \in pre(a) \cup \mathit{eff}(a) \vee \neg f \in pre(a) \cup \mathit{eff}(a)$. We denote the set of actions relevant to agent i as $\mathcal{R}_i(A)$, where $A = \bigcup_{i=1}^{k} A_i$ is the union of the agents' actions.*

Considering these definitions, we can look at the parts of Π and π relevant to agent i. We call this the *local view* of agent i. The part of the MA-STRIPS planning problem relevant to agent i is $\Pi_i = \langle \mathcal{R}_i(F), k, \mathcal{R}_i(A), I, G \rangle$, and the plan π_i is defined as a sequence of joint actions $\mathbf{a}_1^i, \mathbf{a}_2^i, ..., \mathbf{a}_n^i$ where each joint action $\mathbf{a}_t^i$ is a subset of the joint action $\mathbf{a}_t$ that includes only actions $a \in \mathbf{a}_t$ that are relevant to agent i. Actions in $\mathbf{a}_t$ that are not in $\mathcal{R}_i(A)$ are replaced by $\{nop\}$ in $\mathbf{a}_t$.

We can now define the local MA-STRIPS Diagnosis problem for agent i using only its relevant fluents and actions.

▶ **Definition 14 (Local MA-STRIPS Diagnosis Problem).** *A Local MA-STRIPS Diagnosis problem for agent i is a tuple $(\Pi_i, \pi_i, O, M_f, M_c)$ where $\Pi_i = \langle \mathcal{R}_i(F), k, \mathcal{R}_i(A), I, G \rangle$ defines the parts of Π relevant to agent i, π_i is the series of joint actions of plan π that contain only actions relevant to agent i, O_i is the observation received by executing π from I that includes only fluents relevant to agent i, M_f is the fault model and M_c is the conflict model. A local MA-STRIPS diagnosis problem arises when O_i is not consistent with $T_{exp}(I, \pi)$. A diagnosis for that problem is a set of single agent actions $\omega_i = \{a_{j_1}, a_{j_2}, ...\}$ from π_i such that there exists a trajectory $T \in \mathcal{T}(I, \pi, M_c, M_f)$ that is consistent with O_i where $F_i(T) = \omega_i$.*

5.2 Local MA-STRIPS Diagnosis as MBD

Dec-MAS-DX formulates a given local MA-STRIPS diagnosis problem as MBD by specifying SD, COMPS and OBS in a similar way done by MAS-DX, with some important differences.

The first difference is that the components and fluents of the local MA-STRIPS Diagnosis problem for agent i are defined with relation to the local view of agent i. The problem for agent i is modelled as the tuple $(SD_i, OBS_i, COMPS_i)$ where OBS_i are the sets of literals observed in the plan execution that are positive or negative fluents in $\mathcal{R}_i(F)$. The system components are mapped to the actions relevant to agent i: $COMPS_i = \{\{a_{i,t}\}_{i=0}^{k}\}_{t=0}^{n}$ *s.t.* $a_{i,t} \in \mathcal{R}_i(A)$.

The second difference is with the health states defining each component. Here, we need to separate $\mathcal{R}_i(A)$ into two sets - actions that are executed by agent i and actions that are executed by other agents but are relevant to agent i. Formally:

▶ **Definition 15 (Internal Action, External action).** *Given the local diagnosis problem* $(\Pi_i, \pi_i, O, M_f, M_c)$ *for agent* i*, an Internal Action is an action* $a \in \mathcal{R}_i(A)$ *such that* $a \in A_i$*, and external action is an action* $a \in \mathcal{R}_i(A)$ *such that* $a \notin A_i$*. We denote the set of internal and external actions for agent* i *as* $\mathcal{I}_i(A)$ *and* $\mathcal{E}_i(A)$*, respectively. It holds that* $\mathcal{I}_i(A) \cup \mathcal{E}_i(A) = \mathcal{R}_i(A)$ *and* $\mathcal{I}_i(A) \cap \mathcal{E}_i(A) = \emptyset$*.*

The health states for actions in $\mathcal{I}_i(A)$ are defined by the same three health-state predicates as before: $h(\cdot)$, $f(\cdot)$ and $c(\cdot)$. The same predicates cannot be used for actions in $\mathcal{E}_i(A)$. The reason is the fact that such actions are external to agent i, and by definition, only part of the fluents in their preconditions are actually visible to i in its local view (see Definitions 12, 13). To address this, we introduce additional three health state predicates, corresponding to the original three: *external healthy*, *external faulty* and *external conflicted*, and we denote them as $eh(\cdot)$, $ef(\cdot)$ and $ec(\cdot)$, respectively.

Given these changes, SD_i specifies how the fluents of a state change according to the health states of the actions relevant to agent i. In the next section we specify how we encode those changes as SAT boolean formulas.

5.3 Local MA-STRIPS as SAT

5.3.1 Fluents as SAT variables

For every time-step t, we define for each fluent $f \in \mathcal{R}_i(F)$ a boolean variable f_t. This allows us to model every state s as a conjunction of Boolean variables $s_t = \bigwedge_{1 \leq j \leq |\mathcal{R}_i(F)|} f_t^j$. Setting f_t^j to true means that the fluent f^j was part of s_t.

5.3.2 $COMPS_i$ as SAT variables

We model $COMPS_i$ as the set of actions $COMPS_i = \{\{a_{i,t}\}_{i=0}^{k}\}_{t=0}^{n}$ *s.t.* $a_{i,t} \in \mathcal{R}_i(A)$, where k denotes the number of agents and n the length of the plan.

5.3.3 OBS_i as SAT assignments

We encode every observed fluent by setting its corresponding boolean variable value to true or false. Setting f_t^j to true represents that fluent f^j was observed as part of state s_t.

5.3.4 SD_i as a conjunction of SAT formulas

We model SD as a conjunction of Boolean formulas in a similar way we modelled MAS-DX, except for some changes.

For each action $a_{i,t} \in \mathcal{I}_i(A)$, we define how healthy, faulty, and conflicted health states entail the values of fluents in the preconditions of the action. Formally:

11. $h(a_{i,t}) \rightarrow \bigwedge_{f^j \in pre(a_{i,t})} f_t^j$
12. $f(a_{i,t}) \rightarrow \bigwedge_{f^j \in pre(a_{i,t})} f_t^j$
13. $c(a_{i,t}) \rightarrow \neg(\bigwedge_{f^j \in pre(a_{i,t})} f_t^j)$

We do not specify such formulas for the actions $a_{i,t} \in \mathcal{E}_i(A)$, since by definition, the agent does not know the full set $pre(a_{i,t})$ of its external actions. In other words, the variables $eh(a_{i,t})$, $ef(a_{i,t})$ and $ec(a_{i,t})$ do not entail anything about the preconditions of the action.

For each temporal action $a_{i,t} \in \mathcal{I}_i(A)$, the health predicates are mutually exclusive, meaning that exactly one of them must be true. The same is done for actions $a_{i,t} \in \mathcal{E}_i(A)$:

12. $h(a_{i,t}) \otimes f(a_{i,t}) \otimes c(a_{i,t})$
13. $eh(a_{i,t}) \otimes ef(a_{i,t}) \otimes ec(a_{i,t})$

We define the transition formulas similarly to MAS-DX, but with relation to the local view of agent i. In addition, we add transition formulas for the innocent and guilty health states.We begin with transition of variables $f^j \in \mathcal{R}_i(F)$ that are not in any action effects:

14. $\bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \notin \bigcup_{i=1}^{k} \mathit{eff}(a_{i,t})} f^j_{t+1} \leftrightarrow f^j_t$

Values of variables $f^j \in \mathcal{R}_i(F)$ that are in the effects of normal actions are set in the next state to the values according to the actions' effects. Formally:

15. $h(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f^j_{t+1}$

Values of variables included in faulty and conflicted actions are set in the next state to the values according to the respective fault and conflict models. Formally:

16. $f(a_{i,t}) \rightarrow \Phi_f(i,t)$
17. $c(a_{i,t}) \rightarrow \Phi_c(i,t)$

where $\Phi_f(i,t)$ and $\Phi_c(i,t)$ represent the formulas encoding the fault and conflict models M_f and M_c.

We encode similar formulas for actions $a_{i,t} \in \mathcal{E}_i(A)$:

18. $eh(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f^j_{t+1}$
19. $ef(a_{i,t}) \rightarrow \Phi_f(i,t)$
20. $ec(a_{i,t}) \rightarrow \Phi_c(i,t)$

The fault and conflict models are the same as described previously, but here they effect only the relevant fluents.

9. $\Phi_f(i,t) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f_{t+1} \leftrightarrow f_t$
10. $\Phi_c(i,t) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f_{t+1} \leftrightarrow f_t$

Using those formulas, we define the complete SD model:

$$
\begin{aligned}
SD_i = \bigwedge_{t=1}^{n} \Big(\Big(& \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \notin \bigcup_{i=1}^{k} \mathit{eff}(a_{i,t})} f^j_{t+1} \leftrightarrow f^j_t \Big) \\
& \wedge \Big(h(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f^j_{t+1} \Big) \\
& \wedge \Big(f(a_{i,t}) \rightarrow \Phi_f(i,t) \Big) \\
& \wedge \Big(c(a_{i,t}) \rightarrow \Phi_c(i,t) \Big) \\
& \wedge \Big(h(a_{i,t}) \otimes f(a_{i,t}) \otimes c(a_{i,t}) \Big) \\
& \wedge \Big(eh(a_{i,t}) \rightarrow \bigwedge_{f^j \in \mathcal{R}_i(F)\ s.t.\ f^j \in \mathit{eff}(a_{i,t})} f^j_{t+1} \Big) \\
& \wedge \Big(ef(a_{i,t}) \rightarrow \Phi_f(i,t) \Big) \\
& \wedge \Big(ec(a_{i,t}) \rightarrow \Phi_c(i,t) \Big) \\
& \wedge \Big(eh(a_{i,t}) \otimes ef(a_{i,t}) \otimes ec(a_{i,t}) \Big) \Big)
\end{aligned}
$$

Solving this model for agent i will result in sets of health state assignments to the actions relevant to agent i. At this point, some assignments are incorrect due to the partial knowledge that agent i has. The next stage in this approach is to combine the diagnoses of the agents to achieve global diagnoses, which means a set of correct health state assignments that can explain the local observations of all of the agents.

Algorithm 1 computeGlobal.

Input: T - plan size
Input: k - agents number
Input: $\{\mathcal{LD}_i\}_{i=1}^{k}$ - sets of local diagnoses
Result: $\mathcal{GD}$ - a set of global diagnoses
1 sort $\{\mathcal{LD}_i\}_{i=1}^{k}$ by local diagnoses number
2 $\mathcal{GD} \leftarrow \mathcal{LD}_1$
3 **for** $i \in [2, ..., k]$ **do**
4 $\mathcal{NGD} \leftarrow \emptyset$
5 **for** $gd \in \mathcal{GD}$ **do**
6 **for** $ld \in \mathcal{LD}_i$ **do**
7 $ngd \leftarrow combine(k, T, gd, ld)$
8 **if** $ngd \neq None \wedge ngd \notin \mathcal{NGD}$ **then**
9 $\mathcal{NGD} \leftarrow \mathcal{NGD} \cup ngd$
10 $\mathcal{GD} \leftarrow \mathcal{NGD}$
11 **return** $\mathcal{GD}$

5.4 Combining the Diagnoses

At this point, each agent i has a set of local diagnoses $\mathcal{LD}_i = \{ld_{i,1}, ld_{i,2}, ...\}$. Each such local diagnosis contains one assigned health state to each action in $\mathcal{R}_i(A)$. At this stage, some agents might have local diagnoses that disagree about the health state of some actions. The next stage is to combine those local diagnosis sets into one set of global diagnoses, where the agents agree on the health states of the actions.

Algorithm 1 presents this process. The algorithm sorts the agents by the number of local diagnoses from lowest to highest (line 1) and initializes the global diagnosis set as the set of local diagnoses of the first agent (line 2). Then, the algorithm iteratively goes over the agents (Line 3). For every agent, it tries to combine every global diagnosis with every local diagnosis of that agent (Lines 5-7). If the global and local diagnoses agree on their health states, the combination is successful, and the algorithm records the updated diagnosis (Lines 8-9). Finally, the algorithm saves the list of updated global diagnoses (line 10). At the end of the iteration of the last agent, the set of global diagnoses is returned.

▶ **Example 16.** As an example, consider Table 1 again. Let us assume that action 2 of agent 2 ($a_{2,2} = drive(tru2, apt2)$) failed, causing action $a_{2,3}$ to become conflicted. Because of that, agent 0 could not execute action $a_{0,4} = load(apn1, p2)$. Agent 0 can not determine whether $a_{2,3}$ is faulty or conflicted because it does not know about action $a_{2,2}$, which is not relevant to agent 0. Hence, agent 0 outputs two diagnoses. In one, $f(a_{2,3})$ is true, and in another $c(a_{2,3})$ is true. Both diagnoses are inserted into the global diagnoses set to be updated. When these diagnoses are examined by agent 2, it removes the diagnosis where $f(a_{2,3})$ is true. The remaining diagnosis ($c(a_{2,3})$) is corresponding to the local diagnosis of agent 2, where $f(a_{2,2})$ and $c(a_{2,3})$ are true.

Algorithm 2 outlines the process of the combination of two diagnoses. For clarity, we denote the different health predicates as $\mathcal{H}(a_{j,t})$. For example $\mathcal{H}_{ld}(a_{j,t})$ denotes the calculated health state of action a of agent j at time t, according to diagnosis ld. In addition, we extend the equality between health states to include equality between internal and external versions of a health state, for the pairs $h(\cdot)$ and $eh(\cdot)$, $f(\cdot)$ and $ef(\cdot)$, $c(\cdot)$ and $ec(\cdot)$. This way, if an agent diagnosed his internal action $a_{i,t}$ as healthy ($h(\cdot)$), and another agent diagnosed the same (now external from his local view) action as externally healthy ($eh(\cdot)$), then the algorithm will get a false value in line 4.

Algorithm 2 combine.

Input: k - agent number
Input: T - plan length
Input: gd - global diagnosis
Input: ld - local diagnosis
Result: updated global diagnosis
1 $ngd \leftarrow gd$
2 **for** $i \in [1, ..., k]$ **do**
3 **for** $t \in [1, ..., T]$ **do**
4 **if** $\mathcal{H}_{gd}(a_{i,t}) \neq \mathcal{H}_{ld}(a_{i,t})$ **then**
5 **return** None
6 **else if** $\nexists \mathcal{H}_{gd}(a_{i,t}) \wedge \exists \mathcal{H}_{ld}(a_{i,t})$ **then**
7 $\mathcal{H}_{gd}(a_{i,t}) \leftarrow \mathcal{H}_{ld}(a_{i,t})$
8 **return** ngd

The algorithm iterates over all of the agents and time-steps in the plan (Lines 2-3). For every agent and step, in case the health states in the global diagnosis is different than the health state of the local diagnosis, the combining fails, and the algorithm returns no updated global diagnosis (Lines 4-5). Otherwise, if the global diagnosis does not have a health state concerning an action $a_{i,t}$ and the local diagnosis does have it, then the health state of $a_{i,t}$ in the global diagnosis is updated according to the value in the local diagnosis (Lines 6-7). Once the algorithm finishes iterating (Line 8), it returns the updated global diagnosis.

5.5 Proofs of completeness and privacy

We prove completeness by showing that Dec-MAS-DX returns the same set of diagnoses as MAS-DX. We first define the Projected MA-STRIPS Diagnosis problem:

▶ **Definition 17** (Projected MA-STRIPS Diagnosis problem). *Let $DX = (\Pi, \pi, O, M_f, M_c)$ be a given MA-STRIPS diagnosis problem and $Ag \subseteq \{1, \ldots, k\}$ be a set of agents. We define the Projected MA-STRIPS Diagnosis problem Dx_{Ag} as the projection of DX that includes only fluents and actions relevant to at least one of the agents in Ag.*

The problem Dx_{Ag} can be viewed as a diagnosis problem by itself. Hence, we can model it as a SAT model in the same way we model a local MA-STRIPS diagnosis problem. We denote this model as $DX\text{-}Model_{Ag}$. We next prove the following theorem:

▶ **Theorem 18** (Invariant: Consistency of combined diagnosis). *After agent i finishes combining the set of local diagnoses LD_i with the set of global diagnoses GD_{i-1}, the set of global diagnoses GD_i includes the exact diagnosis set received by solving the SAT model $DX\text{-}Model_{1,\ldots,i}$.*

Proof. By induction on the number of agents.

Base (agent 1): For the first agent the diagnoses in LD_1 are inserted directly GD_1. Hence, GD_1 is the set of diagnoses obtained by solving the SAT model $DX\text{-}Model_1$.

Inductive Hypothesis: Assume for agent i.

Inductive Step: Consider agent $i+1$, its set of local diagnoses LD_{i+1} and the set of global diagnoses GD_i received from agent i. Let there be a global diagnosis $gd \in GD_i$ and a local diagnosis $ld \in LD_{i+1}$ that agent $i+1$ currently tries to combine. Consider the health values $\mathcal{H}_{gd}(a_{i',t})$ and $\mathcal{H}_{ld}(a_{i',t})$ of some action for agent i' at some time t according to gd and ld, respectively. We consider three cases:

1. The health variable is $\mathcal{H}(a_{i',t})$ has a value in ld but does not have a value in gd. In this case, action $a_{i',t}$ is not relevant for any of the agents $1, ..., i$, and because of that, the global diagnosis does not keep a health state value. By the inductive hypothesis, that means that in $DX\text{-}Model_{1,...,i}$ there is no variable representing $a_{i',t}$. In the local model of agent $i+1$ there is a variable for action $a_{i',t}$, and there are constraints related to this variable. Hence, when adding the variable and the constraints to $DX\text{-}Model_{1,...,i}$ one possible diagnosis will be gd with the addition of the value of $\mathcal{H}_{ld}(a_{i',t})$ as set in ld. The *Combine* method of Dec-MAS-DX (given in Algorithm 2) returns the same diagnosis.
2. The health value of $\mathcal{H}(a_{i',t})$ in gd and ld is the same. In this case, the existence of a health value for $a_{i',t}$ both in gd and ld implies that the variable for that action exists both in $DX\text{-}Model_{1,...,i}$ and the local model of agent $i+1$. Additionally, $\mathcal{H}_{gd}(a_{i',t}) = \mathcal{H}_{ld}(a_{i',t})$ implies that the health state value for action $a_{i',t}$ in gd is consistent with the constraints in the model of agent $i+1$. This means that if the constraints related to action $a_{i',t}$ in the model of agent $i+1$ are added to $DX\text{-}Model_{1,...,i}$, the diagnosis gd will be generated. In that case, the diagnosis gd is consistent with $DX\text{-}Model_{1,...,i+1}$.
3. The health value of $\mathcal{H}(a_{i',t})$ in gd and ld is not the same. Similarly to case 2, the existence of a health value for $a_{i',t}$ both in gd and ld implies that the variable for that action exists both in $MA\text{-}STRIPS\text{-}DX_{1,...,i}$ and the local model of agent $i+1$. However, $\mathcal{H}_{gd}(a_{i',t}) \neq \mathcal{H}_{ld}(a_{i',t})$ implies that the health value of $a_{i',t}$ that is consistent with the constraints of $MA\text{-}STRIPS\text{-}DX_{1,...,i}$ is not consistent with the constraints of the local model of agent $i+1$ and hence, the set of values in gd combined with the value $\mathcal{H}_{ld}(a_{i',t})$ are not part of a diagnosis, consistent with $MA\text{-}STRIPS\text{-}DX_{1,...,i+1}$. In that case, the combine method of Dec-MAS-DX does not continue with the combination process and returns *none* instead, indicating that the combination of gd and ld is not valid. ◀

▶ **Theorem 19** (Completeness). *Dec-MAS-DX returns all the diagnoses.*

Proof. We prove completeness by showing that Dec-MAS-DX returns the same set of diagnoses as our centralized algorithm, MAS-DX.

From Theorem 18 it follows that when agent k finishes combining the diagnoses, the global diagnoses it returns is the set of diagnoses corresponding to the set of diagnoses returned by solving $DX\text{-}Model_{1,...,k}$, which is the same as the centralized model. Hence, the global diagnoses returned by agent k are the same as the diagnoses returned by MAS-DX. ◀

▶ **Theorem 20** (Privacy). *If a fluent f is only relevant to a single agent, then no other agent can infer its value during the diagnosis process.*

This statement about privacy stems from the fact that the value of this fluent is not in the local view of any other agent. Moreover, whether the value of this fluent is true or false has no direct impact on the applicability or health state of any action of any other agents. Otherwise, it would have been relevant. Note that some fluents that are public, in terms of MA-STRIPS, may still be only relevant to one agent due to the concrete plan the agents were trying to execute.

5.6 Enhancing the Distributed Algorithm

In our Dec-MAS-DX algorithm, once the problem is divided, each local problem has fewer variables in its local MBD formulation but also fewer constraints. This expands the search space, especially for agents that have a lot of external relevant actions.

To address this, we propose an enhanced version of Dec-MAS-DX, which we call Dec-MAS-DX+, in which the agents decide on a computation order based on the upper bound of possible diagnoses of each agent. This number can easily be deduced from the number

of possible health assignments of each agent's relevant actions. If agent i is the one with the lowest bound, it begins its local diagnosis process. Upon finishing, agent i updates the possible health states of its relevant actions. For example, if all of the local diagnoses state that action $a_{i,t}$ is healthy, then agent i updates the possible health values of action $a_{i,t}$ to exclusively healthy. This update includes external actions that are visible to other agents. Then, the remaining agents use the updated possible health states to compute an updated upper bound for the number of diagnoses and decide which agent to perform the local diagnosis next, and so on. At the end of this process, the agents perform the combination process as shown in Algorithm 1, in order to get the final set of global diagnoses.

6 Evaluation

6.1 Experimental Setup

We evaluated our algorithms on eight MA-STRIPS domains *blocksworld*, *depot*, *diverlog*, *logistics*, *rovers*, *satellite*, *taxi* and *zenotravel* taken from the *Competition of Distributed and Multiagent Planners*.[1] For each domain, we experimented with ten problems of different difficulty levels. For each problem, we used an off-the-shelf planning algorithm to generate a plan.[2] Then, for each such problem, we injected $f \in \{1, 2, 3, 4, 5\}$ faulty actions. For each number of faults, we simulated 10 faulty plan executions. For each instance, we observed $o \in \{1, 10, 20, 100\}$ percentage of states, where 1% denotes that we observed the initial and final states, and 100% denotes that we observed every state. In total, we got 2000 instances for every domain. We used the Java Choco library [21] to model the MBD problems as Constraint Satisfaction Problems (CSP) and solve them. We decided to use this library since it is convenient to use and because internally it translates the CSP to SAT.

We then ran each of the algorithms we proposed. Namely, we run the centralized MAS-DX, as well as the distributed algorithms Dec-MAS-DX and Dec-MAS-DX+ (the enhanced version of Dec-MAS-DX that intelligently prioritizes the agents). We limited every instance to run for no more than 10 seconds. We measured the run-time of solving the problems.

6.2 Results

Table 2 presents the average run-time in milliseconds for each algorithm in each domain. We can observe that MAS-DX significantly outperforms in the domains *blocksworld*, *depot*, *rovers*, *taxi* and *zenotravel* (highlighted in bold) and does not outperform in the *driverlog logistics* and *rovers* domains. The reason is that the level of interaction between the actions of the agents differs among the different domains. In the domains where MAS-DX outperforms, the actions of one agent highly interact with the actions of other agents. This is due to many fluents in the plan that are relevant to a number of agents. Therefore, the separation of the central problem to local problems will result in the local problems becoming under-constrained due to many actions becoming external in the local views of the agents. This expands the diagnosis search space, meaning that more local diagnoses will be found. To demonstrate this, we measured the number of local diagnoses found by the agent with the slowest runtime as shown in Table 3. Results show that the decentralized algorithms performed worse in domains with a significantly large number of local diagnoses.

[1] http://agents.fel.cvut.cz/codmap/
[2] https://api.planning.domains/

Table 2 Average run-time in milliseconds of the different algorithms for each domain.

Domain	MAS-DX	Dec-MAS-DX	Dec-MAS-DX+
blocksworld	**31**	4656	3001
depot	**782**	2994	2010
driverlog	72	71	67
logistics00	142	136	**13**
rovers	**18**	1819	1567
satellite	21	11	18
taxi	**12**	2249	119
zenotravel	**7**	3104	2307

Table 3 Average number of local diagnoses computed by the slowest agent in each algorithm.

Domain	MAS-DX	Dec-MAS-DX	Dec-MAS-DX+
blocksworld	3	4721	3275
depot	115	2534	1434
driverlog	14	14	14
logistics00	66	177	7
rovers	3	3350	3057
satellite	1	1	1
taxi	5	4274	148
zenotravel	1	5062	4622

At this stage, we emphasize that the purpose of Dec-MAS-DX and Dec-MAS-DX+ is to address privacy. In both algorithms, each agent is familiar only with the actions relevant to it. We assume that information concerning private actions is not known publicly. Hence, since the process of modeling and solving only accesses the actions of the diagnosing agent and other publicly known actions, no private actions are being shared. This makes the decentralized algorithms relevant to applications where a number of self-interested entities collaborate to achieve a common goal that benefits all of them but where disclosing private information is not desirable.

Another thing to be observed is that Dec-MAS-DX+ significantly improves Dec-MAS-DX in almost every domain. In many cases, distributing the problem leads to uneven distribution of external actions among the agents. This means that for some agents, the local diagnosis space is significantly higher than for others. By solving the local problems for agents with a lower number of external actions first and then using the diagnosis information when solving the local problems of other agents, Dec-MAS-DX+ significantly decreases the run-time of the agent with the largest diagnosis space. This decrease in runtime is so large that it compensates for the serial manner in which Dec-MAS-DX+ works. Specifically in the *logistics* domain, this speedup improvement, combined with the fact that the number of external actions is low, leads to Dec-MAS-DX+ outperforming MAS-DX by an order of magnitude.

7 Conclusions and Future Work

In this work, we presented the problem of MA-STRIPS diagnosis and proposed two approaches for solving the problem. The first approach, MAS-DX, works centrally by modeling the system as a single MBD problem and then solving it. The second approach, Dec-MAS-DX,

addresses privacy requirements by solving a decentralized version of the problem, where agents do not share private information. It first models the local views of the agents to MBD problems of smaller sizes and concurrently solves them, and then computes the global diagnoses by executing a serial process for combining the local diagnoses. Additionally, we proposed an enhancement called Dec-MAS-DX+ that improves Dec-MAS-DX. Empirical evaluation shows that the centralized approach is faster than the distributed approach in most domains, but the benefit of the distributed approach is that it preserves private information.

For future work, we intend to expand our empirical evaluation to other problems and also propose an analysis of the problems in which the distributed approach may run faster. In addition, we aim to generalize our work to multi-agent plans in richer planning formalisms that support stochastic effects and partial observability.

References

1 Craig Boutilier and Ronen I Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001. doi:10.1613/JAIR.740.

2 Ronen Brafman and Uri Zoran. Distributed heuristic forward search for multi-agent systems. In *ICAPS DMAP workshop*, pages 1–6, 2014.

3 Ronen I Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008. URL: http://www.aaai.org/Library/ICAPS/2008/icaps08-004.php.

4 Ronen I Brafman and Carmel Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013. doi:10.1016/J.ARTINT.2012.08.005.

5 Matthew Crosby, Anders Jonsson, and Michael Rovatsos. A single-agent approach to multiagent planning. In *ECAI*, pages 237–242, 2014. doi:10.3233/978-1-61499-419-0-237.

6 Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. Distributed diagnosis of coupled mobile robots. In *Proceedings 2006 IEEE ICRA, 2006.*, pages 3787–3794. IEEE, 2006. doi:10.1109/ROBOT.2006.1642281.

7 Femke De Jonge, Nico Roos, and Cees Witteveen. Primary and secondary diagnosis of multi-agent plan execution. *AAMAS*, 18(2):267–294, 2009. doi:10.1007/S10458-008-9045-X.

8 Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zeev. Diagnosing resource usage failures in multi-agent systems. *ESWA*, 77:44–56, 2017. doi:10.1016/J.ESWA.2017.01.047.

9 Alexander Feldman, Gregory M Provan, and Arjan JC van Gemund. Solving strong-fault diagnostic models by model relaxation. In *IJCAI*, volume 9, pages 785–790, 2009.

10 Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem: A survey. *Satisfiability problem: Theory and applications*, 35:19–152, 1996. doi:10.1090/DIMACS/035/02.

11 Meir Kalech and Gal A Kaminka. Towards model-based diagnosis of coordination failures. In *AAAI*, volume 5, pages 102–107, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-017.php.

12 Meir Kalech and Gal A Kaminka. Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, 27(3):393–421, 2011. doi:10.1111/J.1467-8640.2011.00386.X.

13 Meir Kalech and Avraham Natan. Model-based diagnosis of multi-agent systems: A survey. In *AAAI*, volume 36, pages 12334–12341, 2022. doi:10.1609/AAAI.V36I11.21498.

14 Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving landmark detection. In *ECAI 2014*, pages 597–602. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-597.

15 Shlomi Maliah, Guy Shani, and Roni Stern. Privacy preserving pattern databases. In *Proceedings of the 3rd Distributed and Multiagent Planning (DMAP) Workshop of ICAPS*, volume 15, pages 9–17, 2015.

16 Roberto Micalizio. A distributed control loop for autonomous recovery in a multi-agent plan. In *21st IJCAI*, 2009.

17 Argaman Mordoch, Brendan Juba, and Roni Stern. Learning safe numeric action models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12079–12086, 2023. doi:10.1609/AAAI.V37I10.26424.

18 Avraham Natan and Meir Kalech. Privacy-aware distributed diagnosis of multi-agent plans. *ESWA*, 192:116313, 2022. doi:10.1016/J.ESWA.2021.116313.

19 Tuan Nguyen, Sarath Sreedharan, and Subbarao Kambhampati. Robust planning with incomplete domain models. *Artificial Intelligence*, 245:134–161, 2017. doi:10.1016/J.ARTINT.2016.12.003.

20 Raz Nissim, Ronen I Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1323–1330. International Foundation for Autonomous Agents and Multiagent Systems, 2010. URL: https://dl.acm.org/citation.cfm?id=1838379.

21 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.

22 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

23 Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *AAMAS*, 19(1):30–52, 2009. doi:10.1007/S10458-007-9017-6.

24 Shashank Shekhar and Ronen I Brafman. Representing and planning with interacting actions and privacy. *Artificial Intelligence*, 278:103200, 2020. doi:10.1016/J.ARTINT.2019.103200.

25 Tran Cao Son, William Yeoh, Roni Stern, and Meir Kalech. Multi-agent planning and diagnosis with commonsense reasoning. In *The Fifth International Conference on Distributed Artificial Intelligence*, pages 1–9, 2023. doi:10.1145/3627676.3627690.

26 Michal Štolba and Antonín Komenda. Relaxation heuristics for multiagent planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, pages 298–306, 2014.

27 Michal Štolba, Antonin Komenda, and Daniel Kovacs. Competition of distributed and multiagent planners (codmap). In *AAAI Conference on Artificial Intelligence*, volume 30, 2016.

28 Alejandro Torreno, Oscar Sapena, and Eva Onaindia. Global heuristics for distributed cooperative multi-agent planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 225–233, 2015.

29 Gianluca Torta, Roberto Micalizio, and Samuele Sormano. Explaining failures propagations in the execution of multi-agent temporal plans. In *ICAPS*, pages 2232–2234, 2019. URL: http://dl.acm.org/citation.cfm?id=3332068.

Inferring Sensor Placement Using Critical Pairs and Satisfiability Modulo Theory

Alexander Diedrich
Helmut-Schmidt-University, Hamburg, Germany

René Heesch
Helmut-Schmidt-University, Hamburg, Germany

Marco Bozzano
Fundazione Bruno Kessler, Trento, Italy

Björn Ludwig
Helmut-Schmidt-University, Hamburg, Germany

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy

Oliver Niggemann
Helmut-Schmidt-University, Hamburg, Germany

Abstract

Industrial fault diagnosis exhibits the perennial problem of reasoning with partial and real-valued information. This is mainly due to the fact that in real-world applications, industrial systems are only instrumented insofar, as sensor information is required for their functioning. However, such instrumentation leaves out much information that would be useful for fault diagnosis. This is problematic since consistency-based fault diagnosis uses available information and computes intermediate values within a system description. These values are then used to compare expected normal behaviour to actual observed values. In the past, this was done only for Boolean circuits. Recently, satisfiability modulo non-linear arithmetic (SMT) formulations have been developed that allow the calculation of real values, instead of only Boolean ones. Leveraging those formulations, we in this article present a novel method to infer missing sensor values using an SMT system description and the notion of critical pairs. We show on a running example and also empirically that we can infer novel measurements for five process industrial systems. We conclude that, although SMT calculations accumulate some error, we can infer novel optimal measurements for all systems.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Computing methodologies → Artificial intelligence

Keywords and phrases Sensor Placement, Satisfiability Modulo Theory, Critical Pairs, Diagnosability

Digital Object Identifier 10.4230/OASIcs.DX.2024.9

Funding This research as part of the project EKI is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

Acknowledgements This work has also benefitted from Dagstuhl Seminar 24031 "Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis."

1 Introduction

Consistency-based fault diagnosis identifies faults by comparing predictions from a logical system description to actual observations from a system. In the past, the system description was specified by experts, usually in propositional or predicate logic and often solely for Boolean circuits [39]. It was also assumed that the system description is a complete model



© Alexander Diedrich, René Heesch, Marco Bozzano, Björn Ludwig, Alessandro Cimatti, and Oliver Niggemann;
licensed under Creative Commons License CC-BY 4.0



of the underlying system [9] and thus models all the system's normal behaviour (weak-fault model). However, recently several publications have shown the possibility to learn at least parts of the system description from data for hybrid, i.e. discrete and continous industrial systems [16, 36, 28, 17]. The challenge with those approaches is that they are neither sound nor complete, i.e., they do not guarantee that the complete behaviour of the system is captured. This is due to two reasons: (i) Many of the methods still have severe limitations, (ii) Often observations about the system are missing. Usually, these approaches use a special formulation for the system description of the form

$$\bigwedge_{c_i \in COMPS_k} c_i \rightarrow obs_i \tag{1}$$

for some component $c_i \in COMPS_k$, with $COMPS_k \subset COMPS$ being a subset of the components whose behaviour has an influence on the observables $obs_i \in OBS$. In other words, those approaches reason against the flow of causality. They conclude from anomalous observations back towards possible components causing the anomaly. To obtain Eq. (1) it is necessary to first identify which observables exist within a system (i.e., the system's sensors). Then, system identification and causal reasoning techniques are used to identify those components that have an influence on a single observable. And finally, they use this knowledge to formulate propositional logic system descriptions. The problem with this approach is: since those methods only rely on the observable data, the resulting system descriptions may not be able to distinguish each fault. This means faults in some components may be indistinguishable from each other, thus increasing the diagnostic load of service technicians.

A system description that contains components that are not fully diagnosable suffers from partial observability. Observability can only be increased through the addition of sensors. However, possible locations to add sensors increase exponentially with system size and heavily depend on system complexity. Further, some locations may garner more information than other locations, while some locations may be redundant to already existing sensors.

In this article, we present a solution to infer the placement of new sensors based on three main ideas: (i) For modelling the system, we use the more expressive satisfiability modulo theory with non-linear arithmetic (SMT-$\mathcal{NRA}$) instead of propositional logic. (ii) We compute critical pairs to infer which components are not diagnosable, given a system description, and (iii) we compute the exact locations where to add new sensors as well as the operating range of the sensors' values for non-faulty behaviour. We require the use of SMT-$\mathcal{NRA}$ to be able to compute the non-linear behaviour of most components that are used in industry. Previous work has already shown how SMT-$\mathcal{LRA}$, i.e., SMT using linear arithmetic can be used for fault diagnosis [15]. Critical pairs are a common technique used to identify non-diagnosable components within logical knowledge bases [2]. They rely on system traces, meaning temporally ordered sequences of observations that establish, whether some trace exists that can identify a component unambiguously. By computing critical pairs, we can therefore identify a set of components that is non-diagnosable. Following some of the author's previous work [3] we then show how, given an SMT-$\mathcal{NRA}$ system description, we can compute the possible normal operating ranges of the sensors that would be added for each of the non-diagnosable components. In the end, we can therefore determine which sensors may be added to the system and what the minimum normal operating ranges are. The greatest weakness of our approach is that through the computation of intermediate values through SMT-$\mathcal{NRA}$ significant errors may be introduced depending on the lengths of the paths through the system [18].

Overall, this article makes the following contributions:

- We show how to compute critical pairs based on a system description for consistency-based diagnosis formulated in SMT-$\mathcal{NRA}$.
- We present a novel method to compute normal operating ranges for each possible sensor location as well as minimality criterions for optimal placement.
- We present the novel algorithm SynthesizeObs that first computes critical pairs, then suggests novel observables, and finally selects optimal observables and their normal operating ranges that should be added to an existing system.

With our solution, we thus help practitioners to (i) determine whether their system description is fully diagnosable, and (ii) how to select the optimal sensor placement to increase diagnosability.

2 Running Example

Figures 1 to 3 present the running example that we will use throughout the rest of the article. It is a simple system with three components and two observables.

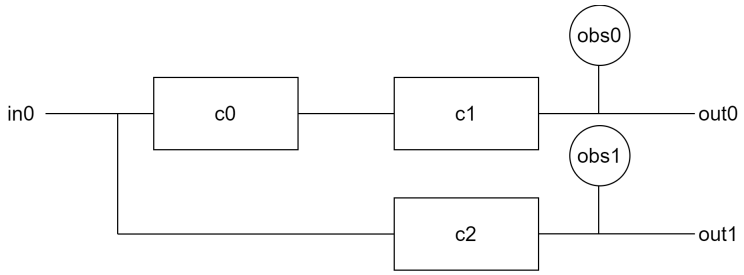


Figure 1 The running example with three components and two observables.

Looking at Figure 1 we assume one input connected to components *c0* and *c2*, with component *c0* connected with *c1*. The outputs *out0* and *out1* of components *c1* and *c2* are monitored by the observables *obs0* and *obs1*. Obviously, since the input, as well as the connection between components *c0* and *c1* is unmonitored, if some fault is recognised in *obs0* both, *c0* or *c1* may be responsible.

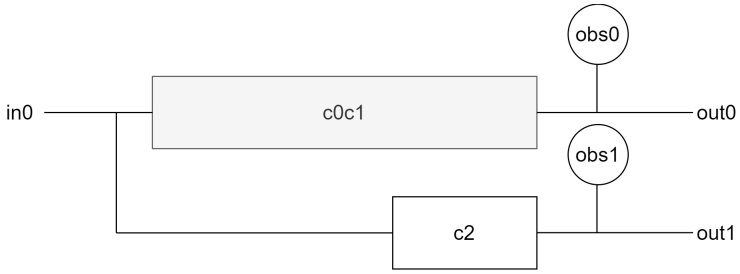


Figure 2 The running example with the virtual component c0c1.

So for practical purposes, *c0* and *c1* may be interpreted as being some virtual component c0c1 as depicted in Figure 2.

Only through the introduction of some additional observation (Figure 3) can the observability be increased and thus faults in *c0* and *c1* be unambiguously distinguished.

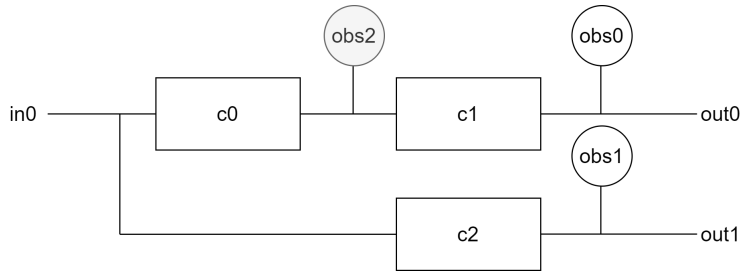


Figure 3 The running example with an additional, synthetically generated, observation to increase the system's observability.

3 Related Work

Within the fields of diagnosability research and fault diagnosis four different categories can be identified: diagnosability of discrete-event systems, analysing hybrid systems (i.e., systems with discrete and continuous behaviour), fault diagnosis itself, and integration with cyber-physical systems. In all of those fields the goal is to investigate whether a model is suitable for diagnosis, while the type of model varies from automata-like approaches, physical equations, logical approaches, and formal specifications.

Diagnosability of discrete-event systems is an important area of research for many academic and industrial use-cases. Usually, those systems are modelled using Petri-Nets [7], automata [33], or similar approaches that divide a systems' operation into states and events. With those modelling formalisms it is necessary that practitioners know which places and states can be visited, whether the model correctly models the real systems under investigation, and, in the case of fault diagnosis, if all faults can be distinguished. One of the first authors who analysed diagnosability of discrete event systems systematically was the approach of Sampath et al. [41] in 1995 that they soon extended with active approaches [40]. A brief history of diagnosability research and its relation to diagnosis is given by Lafortune et al. [30]. More recently, Grastien and others [26, 51] have investigated the diagnosability of discrete-event systems with uncertain observations. Boussif et al. [5] have extended diagnosability analysis to the recognition of intermittent faults with the help of a digital twin model. Bittner et al. [2] have developed a comprehensive work on the formal analysis of discrete-event system diagnosability. From the same research group, Bozzano et al. [6] have investigated the testing for diagnosability. Another recent work has been published by Lamperti et al. [31] on sequence oriented discrete event systems.

Hybrid systems are more complex than discrete event systems. And while in several real world use-cases the continuous behaviour can often be abstracted away or can be roughly approximated, in many other use-cases this is impossible. Therefore, also the diagnosability of models used for hybrid systems needs to be investigated. Benedetto et al. [13] investigated the diagnosability of hybrid automatons. Trave-Massuyes et al. [47] has analysed the diagnosability of analytical redundancy relations and sensor placement. Those relations are common in control theoretic approaches to fault diagnosis (for details, refer [46]). They have also investigated the sensor placement problem [45], which answers the question of where to place sensors within a system to improve diagnosability. Cordier et al. [11] has compared approaches between discrete system diagnosability and approaches to hybrid system diagnosability. The work of Trave-Massuyes was extended first through Nejjari et al. [37] and later through Verdiere et al. [49]. Fourlas et al. [24] were one of the first to specify hybrid system diagnosability in the terminology of discrete-event system diagnosability. Although

this was later taken up by Bayoudh et al. [1] as well. Diene et al. [19] have developed an approach to analyse the diagnosability of linear time invariant systems. Leal et al. [32] have developed an algorithm to check the diagnosability of hybrid systems specified through analytical redundancy relations. Vignolles et al. [50] have analysed diagnosability of hybrid systems with a special emphasis on fault diagnosis and remaining useful life prognosis. Also from a control theoretic standpoint, a recent approach has been introduced by Su at al. [44] where they investigated the sensitivity of residuals with regard to fault isolability. In line with residuals, fault diagnosability was done with structural analysis as described by Krysander et al. [29] and within the seminal work of Escobet et al. [22].

The notion of fault diagnosis itself in this article is in line with the definitions by De Kleer [12] and Reiter [39]. The theory behind fault diagnosis was recently well explained by Feldman et al. [23], Metodi et al. [35], and Stern et al. [43]. Our notion of system descriptions was introduced in earlier work by Diedrich et al. [18] and follows a line of abductive reasoning by Pill et al. [38] and Bochman et al. [4].

In this article, the goal is to apply the diagnosability analysis to system descriptions in order to infer novel observables in cyber-physical systems. In the area of cyber-physical systems research, aspects of these are touched in the verification of cyber-physical system research. Mehrabian et al. [34] have presented an approach for the run-time verification for uncertainties in cyber-physical systems. Caimilli et al. [8] has investigated the verification of cyber-physical systems when they are reconfigured in the presence of faults. Varela-Vaca et al. [48] have proposed a framework for diagnosing and verifying security requirements. While Dowdeswell [20] has shown how to create function blocks for programmable logic controllers' programming that are diagnosable by design.

Summary. Several research fields are concerned with analysing the diagnosability of their respective models. Most important for this article are those using the Dulmage-Mendelsohn decomposition, i.e., approaches that analyse the diagnosability and sensor placement of analytical redundancy relations. This was primarily done by Krysander et al. [29] and Trave-Massuyes [47]. Concretely, we use the formulation of critical pairs from Bittner et al. [3].

4 Background

We work within the setting of standard first-order logic and employ the standard semantic notions of interpretation, model, satisfiability, and validity. A term is defined as either a constant, an individual variable, or an n-ary function symbol applied to n terms. An atom is defined as either a propositional variable or an n-ary predicate applied to n terms. Subsequently, formulae are defined as either atoms, or as the application of the standard Boolean connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ to formulae, or the application of a quantifier $\exists, \forall$ to one or more variables and a formula.

4.1 Logical Modelling

Satisfiability Modulo Theories (SMT) [10] addresses the problem of determining whether a satisfying assignment to the free variables exists in a first-order formula, where non-logical symbols are interpreted within a decidable background theory $\mathcal{T}$.

In first-order logic, an atomic formula is the simplest type of formula that cannot be broken down into smaller logical components and is used to construct more complex formulas.

▶ **Definition 1** (Atomic logical formula). *An atomic logical formula is an expression consisting of one predicate applied to terms and has a Boolean truth value in* $\{\top, \bot\}$.

Within this work, we focus on the theory of *Nonlinear Arithmetic over the reals* ($\mathcal{NRA}$). The theory of $\mathcal{NRA}$ is the first-order theory with equality whose atoms are nonlinear polynomial constraints interpreted over $\mathbb{R}$. We have numerical constants and real-valued variables. Using the standard function symbols $+, -, \cdot, \div$, these numerical constants and real-valued variables can be combined to form arithmetic terms.

▶ **Definition 2** (Arithmetic term). *An arithmetic term in SMT is an expression consisting of a variable, a numerical constant, or combinations thereof connected through* $\{\cdot, +, -, \div\}$.

With the predicate symbols $<, \leq, >, \geq, =$, arithmetic terms can be combined with numerical constants or other arithmetic terms to form arithmetic formulae. Depending on the number of combinations, the resulting formula can be an atomic arithmetic formula.

▶ **Definition 3** (Atomic arithmetic formula). *An atomic arithmetic formula is an expression consisting of two arithmetic terms connected through* $\{<, \leq, >, \geq, =\}$ *and has a Boolean truth value in* $\{\top, \bot\}$.

SMT allows the combination of first-order logical and theory specific formulas with the logical connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ in order to express complex contexts.

▶ **Definition 4** (SMT-$\mathcal{NRA}$ formula). *An SMT-$\mathcal{NRA}$ formula denotes the combination of (multiple) atomic first-order logical formulas and (multiple) atomic arithmetic formulas, connected via the logical connectives* $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

If φ_0, φ_1 are atomic formulae or SMT formulae, then $\varphi = \varphi_0 \oplus \varphi_1$ with $\oplus \in \{\wedge, \vee, \rightarrow, \leftrightarrow, =, \leq, \geq\}$ is also an SMT formula. This SMT formula φ is considered satisfiable if and only if it has a model M.

▶ **Definition 5** (SMT Model). *A model M of a formula φ is defined as a pair consisting of an assignment, which maps each variable to an element of its domain, and an interpretation of the non-logical symbols that satisfies the formula.*

▶ **Definition 6** (Satisfiability). *An formula φ is satisfiable, when a model M exists, where* $[[\bigwedge_{\varphi_i \in \varphi} \varphi_i]]_M = \top$.

The satisfiability of an SMT formula φ can be checked by an SMT solver. An SMT solver is a software tool designed to determine whether a given formula is satisfiable; it may either provide a satisfying assignment or a proof of unsatisfiability. Most modern SMT solvers employ a lazy approach: they first use a SAT solver to enumerate potential satisfying assignments of the formula's Boolean abstraction. Then, theory solvers verify the satisfiability of the corresponding set of constraints for each assignment. If the theory solver finds the set of constraints consistent, a model is established. Otherwise, new clauses are generated to prevent the recurrence of assignments that lead to inconsistencies.

Especially in the context of fault diagnosis, often it is not only relevant to determine the satisfiability of an SMT formulae or the proof of unsatisfiability, but also to find the maximum satisfiability set of a formula φ that is unsatisfiable.

▶ **Definition 7** (Maximum Satisfiability). *A maximum satisfiable set (mss) of a set of SMT expressions φ are those expressions that can be satisfied even though φ itself is unsatisfiable, with $mss = argmax_{\alpha,M}\varphi$, where α is a complete assignment to all variables within φ, using a single model M.*

4.2 Fault Diagnosis

This section presents the theoretical background on i) how to obtain observations OBS : $\mathbb{R}^{n\times m}, n, m \in \mathbb{N}$ from some system, and ii) how a given system description can be used to perform diagnosis.

We begin by defining a single measured value from a cyber-physical system as a sensor reading $m_i \in \mathbb{R}$. The explicit mention of time is omitted, as we assume that all sensor values are read concurrently at a single point in time, given a suitable sampling rate. Therefore, we write m_i instead of $m_i(t)$. Observations are discretised sensor readings that we define by some suitable function $d(\cdot)$.

▶ **Definition 8** (Observation). *A single observation* $obs \in \{\top, \bot\}$ *is a discretised assignment* $obs_i = d(m_i, \boldsymbol{\lambda})$ *to some or all inputs and outputs of a cyber-physical system, where* $d(\cdot)$*, with* $d : \mathbb{R} \to \{\top, \bot\}$*, is a suitable discretisation function,* $\boldsymbol{\lambda} \in \mathbb{R}^{|\lambda|}$ *are parameters, and* $m_i \in \mathbb{R}$*.*

We will write $|\lambda|$ to denote the cardinality of $\boldsymbol{\lambda}$. The set of all observations is denoted as OBS. In other words, we require the existence of a function $d(\cdot)$ that discretises a real-valued input with the parameters $\boldsymbol{\lambda}$ into a Boolean value. Usually, function $d(\cdot)$ is realised through residual values [27]. A residual takes some of the observable parameters of a cyber-physical system in the form of input, output, and state variables and outputs 0 in the fault-free case and $\neq 0$, otherwise. Formally, a residual is a function

$$r(m_i, \boldsymbol{\lambda}) = \begin{cases} 0; \text{ no fault} \\ \neq 0; \text{ else} \end{cases} \tag{2}$$

with sensor value $m_i \in \mathbb{R}$ and $\boldsymbol{\lambda} \in \mathbb{R}^{|\lambda|}$ is a set of parameters. Usually, the set of parameters $\boldsymbol{\lambda}$ contains guard conditions around the input, output, and state variables. The set of all residuals is denoted as $\mathcal{R}$. Several existing approaches use observer or structural equations to calculate residual values, but rely on expert-defined first-principles models. [25] provide a good overview over existing methods to generate residuals.

Example. An implementation of the discretization function in the form of a residual can be done through, for example, threshold values $\boldsymbol{\tau}$, such that $d(m_i, \tau) : \tau_l \leq m_i \leq \tau_h \to obs_i$, where $\tau_l, \tau_h \in \mathbb{R}$ are some threshold values, obs_i is a symbol denoting the observation, and $m_i \in \mathbb{R}$ is the denoised sensor value. In previous work, we showed that such threshold equations can be translated into propositional logic through satisfiability modulo linear arithmetic theory [15].

With residuals, real-valued signals from a cyber-physical system are discretised into Boolean observations. But how is it possible to use these for fault diagnosis? Fault diagnosis algorithms are employed to identify inconsistencies between the observations from a cyber-physical system and its system description. Formally, a diagnostic system is defined as follows.

▶ **Definition 9** (Diagnostic System). *A diagnostic system is a three-tuple* (SD, COMPS, OBS)*, where* SD *is the system description formulated in propositional logic,* COMPS *is the set of components, and* OBS *are the Boolean observations grounded as propositional symbols, such that* SD ∪ COMPS ∪ OBS *is satisfiable.*

In order to perform fault diagnosis, non-monotonic and abductive reasoning are employed. This allows the system description to reason with an inconsistent knowledge base and conclude from observations to causes. Reiter [39] called non-monotonic reasoning default

reasoning whose main benefit is that it contains logical statements that are assumed to be true unless proof to the contrary exists. Diagnosis uses non-monotonic reasoning to retract those assumptions within SD (i.e. those components that might have caused some observation to deviate from its normal behaviour) until a consistent diagnostic system is achieved. From those retracted assumptions the diagnosis is then computed.

▶ **Definition 10** (Diagnosis). *A diagnosis ω is a set derived from the set of retracted assumptions, which satisfies* $\text{SD} \wedge \text{OBS} \wedge \text{COMPS}_{\omega} \wedge \text{COMPS}_{\neg\omega}$

where COMPS_{ω} are the faulty components and the set $\text{COMPS}_{\neg\omega}$ are the remaining healthy components. Within this article we use a compilation-based approach and compute the diagnosis by calculating the maximum satisfiable set within an SMT-$\mathcal{NRA}$ system description. In many cases, the size of individual diagnoses can be quite large and sometimes contain hundreds of components. Therefore, the size of a diagnosis needs to be limited. A minimum cardinality diagnosis is a diagnosis that contains the smallest possible number of components.

▶ **Definition 11** (Minimum Cardinality Diagnosis). *A Minimum Cardinality Diagnosis is a diagnosis ω', so that $|\omega'| \leq |\omega|$ for each diagnosis.*

Whenever we employ a diagnosis algorithm, the goal is always to find the minimum cardinality diagnosis.

5 Problem Description

To diagnose faults in technical systems, a system description is required that describes the system's normal behaviour. For this work, we assume the existence of an SMT-$\mathcal{NRA}$ system description. The system description is divided into two parts: The first part describes each component's behaviour and the second part describes the observables, modelling the evaluation of each observable value. More formally, we divide SD into $SD = \mathcal{B} \cup \mathcal{O}$, for the SMT-$\mathcal{NRA}$ behavior description $\mathcal{B}$ and the observable description $\mathcal{O}$

For modelling $\mathcal{B}$ we extend the formalism that was originally suggested by Feldman et al. [23] for propositional logic

$$h_i \rightarrow out_i = g_i(\mathbf{in}_i) \tag{3}$$

where $h_i \in \{\top, \bot\}$ indicates the health of the respective component with index $i \in \mathbb{N}$. The function $g_i(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}$ computes the output $out_i \in \mathbb{R}$ from inputs $\mathbf{in}_i \in \mathbb{R}^n$, with index $i \in \mathbb{N}$ and number of input signals $n \in \mathbb{N}$. Overall, Equation (3) is said to be a weak-fault model of the system, i.e., it describes only the system's normal behaviour.

To observe whether the system behaves normally, we need to add the second part of the system description $\mathcal{O}$, using the observables $obs \in OBS$. We therefore use expressions of the form

$$h_i \rightarrow obs_i = d(out_i, \theta_i) \tag{4}$$

with observable $obs_i \in \{\top, \bot\}$, input $out_i \in \mathbb{R}$, a set of parameters $\theta \in \mathbb{R}^n$, and an observation evaluation function $d(\cdot) : \mathbb{R} \rightarrow \{\top, \bot\}$, where $h_i \in \{\top, \bot\}$ indicates the health of the sensor.

The problem we tackle in this article is two-fold: (i) We investigate which components within a given system description SD can be diagnosed given the available observations. In other words, which observations will allow us to diagnose a fault in a component unambiguously, such that only the real faulty component is identified? For this we use a single-fault

assumption such that we assume only one component is faulty at a given time. (ii) Given a set of non-diagnosable components, which observables do we need to increase the system's observability and decrease the set of non-diagnosable components?

To illustrate the problems we use the running example with an exemplary and simplified system descriptions such as

$$h_0 \rightarrow int_0 = g_0(in_0) \tag{5}$$
$$h_1 \rightarrow out_0 = g_1(int_0) \tag{6}$$
$$h_2 \rightarrow out_1 = g_2(in_0) \tag{7}$$
$$h_a \rightarrow obs_0 = d(out_0, \theta_0) \tag{8}$$
$$h_b \rightarrow obs_1 = d(out_1, \theta_0) \tag{9}$$

We can now assume concrete realisations of functions g, d. For this, lets assume in the running example we have a process industrial system with a pump c_0, and valves c_1, c_2.

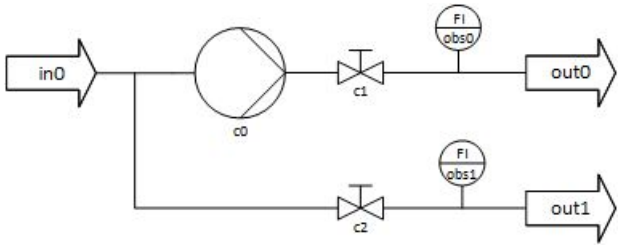


Figure 4 The running example using real physical components.

Using this, we can specify the physical modelling functions $g(\cdot)$ as part of the behaviour system description $\mathcal{B}$ through the equations for tanks and valves. For pumps, we use the model

$$\Delta P = P_{max} - R_p Q \tag{10}$$

where $Q \in \mathbb{R}$ is the flow rate, $\Delta P \in \mathbb{R}$ is the pressure difference, $P_{max} \in \mathbb{R}$ is the maximum pressure the pump can generate, and $R_p \in \mathbb{R}$ is the pump resistance. For a valve, we can for simplicity assume the following model

$$Q = Cx\Delta P \tag{11}$$

where $Q \in \mathbb{R}$ is the flow rate, $\Delta P \in \mathbb{R}$ is the pressure difference, $C \in \mathbb{R}$ is the valve coefficient, and $x \in [0 \dots 1]$ is the valve opening.

We then model the observables within part $\mathcal{O}$ of the system description through

$$obs_i = \tau_l \leq out_i \leq \tau_h \tag{12}$$

with the thresholds $\tau_l, \tau_h \in \theta_i$.

Overall, we can now specify the complete system description as SMT-$\mathcal{NRA}$ expressions

$$\text{Pump } c_0\text{: } h_0 \rightarrow int_0 = P_{max} - R_p in_0 \tag{13}$$
$$\text{Valve } c_1\text{: } h_1 \rightarrow out_0 = Cx\Delta int_0 \tag{14}$$
$$\text{Valve } c_2\text{: } h_2 \rightarrow out_1 = Cx\Delta in_0 \tag{15}$$
$$\text{Observable } obs_0\text{: } h_a \rightarrow obs_0 = \tau_{l_0} \leq out_0 \leq \tau_{h_0} \tag{16}$$
$$\text{Observable } obs_1\text{: } h_b \rightarrow obs_1 = \tau_{l_1} \leq out_1 \leq \tau_{h_1} \tag{17}$$

It is evident that by placement of the observables component c_0 (the pump) cannot be diagnosed. In what follows we present first a method that establishes the diagnosability of systems using these types of system descriptions and then, in section 7, we will describe how to infer additional sensor values to increase a system's diagnosability.

6 Finding Critical Pairs for Diagnosability in SMT-$\mathcal{LRA}$ System Descriptions

Historically, diagnosability can be reduced to the search for critical pairs [3, 2]. A critical pair is defined based on the definition of traces (formalizing an execution of the plant).

▶ **Definition 12** (Plant). *A plant P is a* symbolic transition system $\langle V, V_o, I, T\rangle$*, where V is a finite set of state variables; $V_o \subseteq V$ is the set of observable state variables; I is a formula over V defining the initial states, and T is a formula over V, V' (with V' being the next version of the state variables) defining the transition relation.*

A *state* s is an assignment to the state variables V. The *observable part* $obs(s)$ of a state s is the projection of s on the subset V_o of observable state variables.

▶ **Definition 13** (Trace). *A system trace is an infinite sequence of states $s_0, s_1, s_2, \ldots$ starting from s_0 such that, for each $k \geq 0$, $\langle s_k, s'_{k+1}\rangle$ satisfies T.*

The observable part of a trace $\pi = s_0, s_1, s_2, \ldots$, denoted $obs(\pi)$, is the sequence $obs(s_0)$, $obs(s_1)$, $obs(s_2)$, ...

In this paper, we are interested in diagnosing the faulty status of components, i.e., a diagnosis for component c can be defined as the healthy status h_i of the i-th component. We give the following definition for critical pairs.

▶ **Definition 14** (Critical Pair). *Given a plant P, a diagnosis condition c and a delay d, a critical pair for P and c at i is a pair of traces π_1 and π_2 of length $i + d$ such that $obs(\pi_1) = obs(\pi_2)$ (traces are observationally equivalent), $\pi_1, i \models c$ (trace π_1 satisfies c at i), and $\pi_2, j \not\models c$ for all j such that $j \in [i - d, i + d]$ (trace π_2 does not satisfy c in the interval $[i - d, i + d]$).*

Our definition of diagnosability corresponds to the notion of *bounded delay diagnosability*, i.e., diagnosability within a delay d, following [2].

Searching for critical pairs can be done using the twin-plant construction [2], and can be implemented using an SMT solver, by unrolling the transition relation and looking for a pair of traces satisfying the conditions of Def. 14.

7 A Novel Method to Synthesize new Observables

We have seen in the last section how critical pairs identify those components that, given two observable traces, cannot be distinguished in terms of their faulty behaviour. In this section, we describe a novel method to synthesize new minimal observables, given the SMT-$\mathcal{NRA}$ system description SD and a set of critical pairs.

The intuition behind our approach is the following: We assume that there exists an SMT-$\mathcal{NRA}$ system description. Then we first compute the intermediate values (assumables) that are the output of the non-observable component through logical inference. For each of those outputs we then attempt to compute a new observable that is described through a logical expression template and its historical values.

We therefore use the following definition.

▶ **Definition 15** (Novel observable). *A novel observable is an additional measurement* $obs_i \in \mathbb{R}$ *with an associated range* $r_l, r_h \in \mathcal{R}$ *with* $r_l \leq obs_i \leq r_h$ *during normal operating conditions.*

we say that the range of a variable is defined through the two functions

$$obs_{i_l} = \underset{in_i}{\operatorname{argmin}}\, g(in_i, \theta_i) \tag{18}$$

and

$$obs_{i_h} = \underset{in_i}{\operatorname{argmax}}\, g(in_i, \theta_i) \tag{19}$$

where argmin and argmin take the ranges of the normal operating conditions, given by the underlying system. In other words, the values of in_i are constrained by the values seen in historical data under normal working conditions. The calculation of these intermediate values is easy, since we use SMT-$\mathcal{NRA}$ for the behaviour models of all components. Thus, we can calculate the outputs of each component up to an accuracy limited by the SMT solver.

To combat the accumulation of errors in the sequential computation of ranges [18] we introduce several heuristics. The first is the range-minimal novel observable.

▶ **Definition 16** (Range-minimal Novel observable). *A range-minimal novel observable is an additional measurement* $obs_i \in \mathbb{R}$ *with additional constraints limiting the range* $r_l, r_h \in \mathcal{R}$ *with* $r_l \leq obs_i \leq r_h$*.*

We use this observable to obtain those novel observables that have the smallest normal operating range and thus are assumed to be the most sensitive to faults. In other words, we want those observables whose range does not include too much operational variability.

Another helpful definition is the count-minimal novel observable. Using these observables the highest number of critical pairs can be removed and the maximum number of components become diagnosable.

▶ **Definition 17** (Count-minimal Novel observable). *A Count-minimal novel observable is an additional measurement* $obs_i \in \mathbb{R}$ *with range* $r_l, r_h \in \mathcal{R}$ *with* $r_l \leq obs_i \leq r_h$ *that removes the maximum number of critical pairs.*

The range- and count-minimal novel observable can be determined using weak-fault models, i.e., models only describing the normal working behaviour of the system [23]. These are always calculated first. It is then possible to apply heuristics to these novel observables in order to improve the diagnostic performance.

To obtain heuristics, we require more parameters than a common weak-fault model provides. After having calculated the range- or count-minimal observables we can thus calculate those observables that provide the maximum information gain. The intuition is as follows: Assuming we want to add those observables that provide the maximum information gain, we first need to determine how much more we can diagnose, if specific sensors are added. In earlier work, we demonstrated that such information can be obtained using entropy and information-gain calculations [14]. To calculate the entropy we require information about the reliability of single components which usually can be obtained either from experience or through design documentation such as mean-time-between-failure analysis.

▶ **Definition 18** (Entropy-maximal Novel observable). *An entropy-maximal novel observable is an additional measurement* $obs_i \in \mathbb{R}$ *with range* $r_l, r_h \in \mathcal{R}$ *with* $r_l \leq obs_i \leq r_h$ *whose value adds the maximal amount of knowledge about the system.*

We suggest an entropy-based approach [42] to reduce critical pairs by introducing an entropy-maximal novel observable. An entropy-based approach facilitates that an observable leads to a maximal information gain. We assume that the set of observables OBS can be divided into two subsets OBS_u, OBS_c, with $OBS_u \cap OBS_c = \emptyset$, and $OBS_u \cup OBS_c = OBS$, where $OBS_u = \{obs_0, obs_1, ..., obs_{n-1}\}, n \in \mathbb{N}$ is the set of sensors currently used to diagnose the system and $OBS_c = \{obs_{n+1}, obs_{n+2}, ..., obs_m\}, n, m \in \mathbb{N}$ is the set of candidate sensors that could be added to the system (i.e. the determined range-minimal, or count-minimal observables). The intuition now is to calculate the entropy based on the current sensor setup OBS_u, reflecting the uncertainty in distinguishing between different states given the existing sensors. For each sensor of the set OBS_c calculate the conditional entropy to see how much uncertainty remains if that sensor is added.

The entropy is calculated through

$$H(X) = -\sum_{x \in X} p_{OBS_c}(y) \log\left(p_{OBS_c}(y)\right) \tag{20}$$

with $x \in X$ describing the different possible states of the system's components and $p_{OBS_c}(y)$ describing the component's probability of being in that faulty state, under the condition that the component is not observable.

We can then calculate the information gain for each possible observable placement through the individual information gain

▶ **Definition 19** (Information Gain). *The information gain by adding a sensor $obs_i \in OBS_c$ to the set OBS_u is defined over the given entropy H as follows:*

$$IG_H(obs_i) = H(X) - H(X|obs_i) \tag{21}$$

Note that $H(X|obs_i) \leq H(X)$ resulting in $IG_H(obs_i) \geq 0$ and that maximizing the information gain $\max_{s_i \in \mathcal{S}_c} IG_H(s_i)$ can be directly reformulated as a minimization problem of the form $\min_{s_i \in \mathcal{S}_c} H(X|s_i)$.

To illustrate, consider equipping the running example shown in Figure 3 with novel observables: First we have to determine all ways in which the system may fail. There are multiple cases that could lead to faulty outputs. Overall, faults within c_0, c_1, and c_2 might cause one or both outputs to be incorrect. These possible states are detailed in Table 1. The last column of this table lists the minimal hitting sets, indicating which components, if faulty, would result in the observed output.

Assuming a fictitious fault probability of five percent for all components, the entropy values and information gains are presented in Table 2. The first column provides the index, while the second column lists the possible diagnosis in terms of single faults. Subsequent columns show the entropy values for the current set of sensors and for scenarios with one additional observable. Table 3 presents the entropy value $H()$ for each index and the corresponding information gain $IG_H()$. The optimal observable to place is the one that minimizes entropy or maximizes information gain. In this case, the decision is not straightforward, as obs_0 and obs_1 both yield the same information gain.

7.1 Algorithm

Algorithm SynthesizeObs summarises the above definitions into pseudocode. The intuition behind the algorithm is to detect non-observable components using the calculation of critical pairs. Once the critical pairs within a system are known, we calculate optimal observables.

Table 1 Different states of the running example and possible diagnosis.

Index	out_0	out_1	ω_{min}
0	0	0	-
1	1	0	$c_0 \vee c_1$
2	1	1	$(c_0 \vee c_1 \vee) \wedge c_2$
3	0	1	c_2

Table 2 Entropy for all possible observations.

Index	$\omega_{min} \cap \mathcal{S}_c$	$H(x)$	$H(X\|c_0)$	$H(X\|c_1)$	$H(X\|c_2)$
1	$c_0 \vee c_1$	0.09860	0.0629	0.0629	0.09860
2	$(c_0 \vee c_1) \wedge c_2$	0.0108	0.006	0.006	0.09860
3	c_2	0.0629	0.0629	0.0629	0

Table 3 Entropy and Information Gain for possible new observables.

	X	$X\|obs_0$	$X\|obs_1$	$X\|obs_2$
$H()$	0.1723	0.1318	0.1318	0.1972
$IG_H()$	0	0.0405	0.0405	-0.0249

If information about failure rates is available, we use it to calculate the entropy-maximal observables. If no further information is available, we return either the count-minimal, or the range-minimal novel observable, depending on the concrete implementation.

We will now explain the algorithm in detail. In Line 3 we use the SMT system description and the current observations to infer all intermediate values within the system using an appropriate solver for SMT-$\mathcal{NRA}$. This is basically how GDE [12] works on propositional logic. Line 4 calculates the critical pairs from the system description by generating system traces using the available observations. If the set of critical pairs is non-empty, Line 6 generates either a count-minimal, or range-minimal novel observable through properly designing function GENERATEOBSERVABLE. The function determines the ranges from the inferred values $\mathcal{I}$ of the system description, and through the historical observations $HIST$. Line 7 determines whether failure probabilities are known for each component. If this is not the case the algorithm only returns the range-minimal or count-minimal novel observable. Contrarily, if the failure information is available, the algorithm computes the entropy of possible novel observables and their information gain (Line 9). All possible novel observables are added to set $\mathcal{C}$. Line 11 finally only returns the entropy-maximal novel observable.

8 Evaluation

To evaluate our approach, we have used the Berfipl benchmark with four tank systems developed by Ehrhardt et al. [21]. The benchmark describes an industrial bottling plant and was implemented as a simulation using OpenModelica 1.13. The four modules are a mixer, a distill, a filter, and a bottling module. The modules may be used stand-alone or may be connected. For the evaluation in this article, it was assumed that the modules are stand-alone and effects from one module do not propagate into another module. The mixer perturbs three substances and stores the mixed products in a tank. The mixer is connected to a distill, which splits substances with different boiling temperatures. The filter subsystem

Algorithm 1 `SynthesizeObs`: Calculate synthetic novel observables.

Data: SD, OBS, $HIST$
Result: Entropy-maximal observable

$\mathcal{C} \leftarrow \emptyset$;
$\mathcal{I} \leftarrow inferValues(SD, OBS)$;
for $c_i \in criticalPairs(SD, OBS)$ **do**
  $obs_{new} \leftarrow generateObservable(c_i, \mathcal{I}, HIST)$;
  if $\forall c_i, p(c_i)$ *is known* **then**
    $H(X) = -\sum_{x \in X} p(obs_i) \log\left(p(obs_i)\right)$;
    $IG_H(obs_i) \leftarrow H(X) - H(X|obs_i)$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup (obs_{new}, IG_H(obs_i)$;
  else
    $\mathcal{C} \leftarrow \mathcal{C} \cup obs_{new}$;
  $return(max(\mathcal{C})$;

removes residue from the fluid stream. As the last module, the bottling subsystem fills the produced substance into tiny glass bottles. Figure 5 shows the piping and instrumentation diagram of the four modules: a) filter, b) mixer, c) distill, and d) bottling.

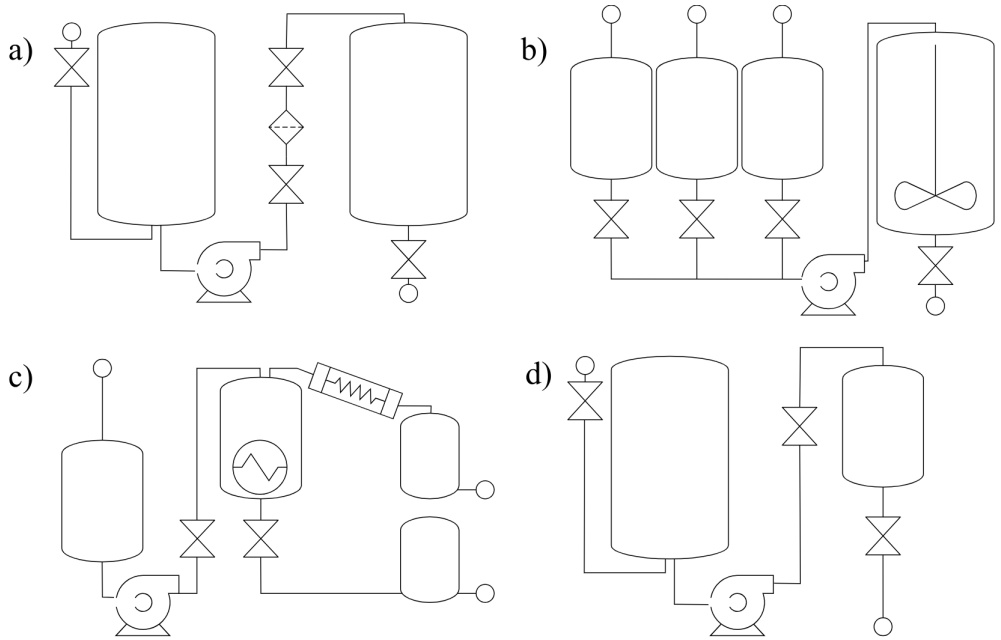


Figure 5 Four modules of the Berfipl benchmark presented by Ehrhardt et. al [21].

For the evaluation we have used an abstract model of each subsystem implemented in Python 3.11 and run on a 64-bit Windows 11 operating system with 32GB RAM, and Intel Core i7-12800H processor. For simplicity, and to keep the number of SMT expressions small we only calculate the flow through all components, while we did not model the intricacies of the special components filter and mixer. We modelled valves as a constant parameter that adjust the flow rate by multiplication

$$outflow = C \times a \times inflow \tag{22}$$

Table 4 Experimental results for the Berfipl benchmark [21].

System	Critical Pair	Range	Correct
DS1	pump0, valve1	[0..9]	✓
	valve1, filter0	[0..9]	✓
	filter0, valve2	[0..9]	✓
DS2	valve0, pump0	[0..6.72]	✓
	valve1, pump0	[0..6.72]	✓
	valve3, pump0	[0..6.72]	✓
DS3	pump0, valve1	[0..9]	✓
DS4	pump0, valve1	[0..9]	✓

with parameter $C \in \mathbb{R}$, the size of the opening $a \in [0..1]$, and the inflow $inflow \in \mathcal{R}$. In the same way we have modelled pumps such that they increase the flow, using

$$outflow = P_max - Rp \times inflow \tag{23}$$

with $P_max, Rp \in \mathbb{R}$ modelling the pressure drop within the pump. Finally we have modelled tanks through

$$outflow = C \times a \times \sqrt{2g(level + inflow)} \tag{24}$$

with parameter $C \in \mathbb{R}$, size of the output orifice $a \in \mathbb{R}$, the gravitational constant g, and sum of the tank level and inflow. We then ran algorithm SynthesizeObs on each of the model, identified the critical pairs and calculated the range-maximal novel observations for each possible sensor location.

Table 4 presents the empirical results of algorithm SynthesizeObs. Since we have used the same parameters for the tanks, pipes, pumps etc. the different ranges are comparable. For each experiment, the algorithm was able to calculate the correct ranges for potential synthetic observations. Limitations are evident that by using SMT-$\mathcal{NRA}$ we calculate the square root within the tank equations. Here we observed errors up to five percent.

We have also evaluated the generation of entropy-maximal novel observables. For this evaluation, we have used a slightly larger system than the running example. Consider the system in figure 6 that contains a source, two tanks and a sink. The pipes between source and tank1 as well as between the two tank are equipped with a pump and a valve each. Consider the system to be equipped with one observable per tank that measures the fill level and provides two observations: high-level and low-level. The possible states of the observables can be seen in Table 1. The pumps can have two fault modes: higher p_xh or lower p_xl performance than expected. The valve may fail through clogging v_x. The possible faults are summarised in the last column as single faults. Possible combinations are not shown.

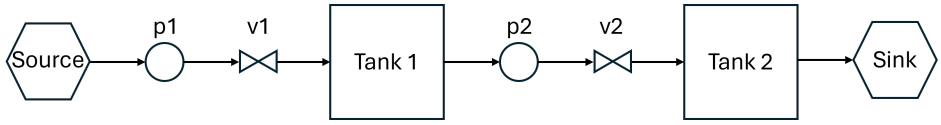


Figure 6 Example tank system.

Some faults have just one single element as a possible diagnosis. These failure states can be directly diagnosed. Others show multiple components that could cause this effect. However, given the observables, in many cases the diagnosis can contain multiple components.

The problem of critical pairs arise if two components cannot be distinguished. This is the case for the combination of pump and valve in each pipe.
If we let the probability of a fault in the pumps and valves be five percent this leads to the entropy values and information gain stated in Table 5. The first column denotes the index. The second column shows the possible diagnosis. The following columns show the entropy values for the current sensors and one additional sensor. The lower table shows the entropy value $H()$ for each index and the information gain $IG_H()$. The best sensor to place is the one that results in the minimum entropy or gives the maximum information gain. In this case this is sensor p_2.

Table 5 Possible states of the system.

State ID x	$\omega_{min} \cap \mathcal{S}_c$	$H(x)$	$H(x\|p_1)$	$H(X\|v_1)$	$H(X\|p_2)$	$H(X\|v_2)$
1	$p_1h \vee p_2l \vee v_2$	0.195	0.13	0.195	0.13	0.13
2	p_1h	0	0	0	0	0
3	$p_2l \vee v_2$	0.13	0.13	0.13	0	0
4	$p_1l \vee v_1 \vee p_2h$	0.195	0.13	0.13	0.13	0.195
5	p_2h	0	0	0	0	0
6	$p_1l \vee v_1$	0.13	0	0	0.13	0.13
7	p_2h	0	0	0	0	0
8	$p_2l \vee v_2$	0.13	0.13	0.13	0	0

	X	$X\|p_1$	$X\|v_1$	$X\|p_2$	$X\|v_2$
$H()$	0.78	0.52	0.585	0.39	0.455
$IG_H()$	0	0.26	0.195	0.39	0.325

9 Discussion and Conclusion

While the evaluation has shown that we can successfully infer range- and count-minimal novel observations using SMT-$\mathcal{NRA}$ system descriptions and the computation of critical pairs, our method has some practical drawbacks:

1. Using SMT-$\mathcal{NRA}$ for process industrial systems requires the computation of square roots. But computing the square roots introduces significant errors through chaining effects.
2. The SMT-$\mathcal{NRA}$ system descriptions we use are quite simple. Computing critical pairs on those system descriptions does not distinguish between the type of component. In practice, some components, such as a filter will lead to different downstream effects than, for example, a tank. Therefore, some components and their effects may be diagnosable, even though there is no sensor directly attached to them.
3. We performed the entropy calculations outside of the SMT-$\mathcal{NRA}$ system description. In future work this should be integrated to obtain a unified formulation for calculating novel observables.
4. Our approach requires significantly more expert knowledge than is available for many industrial processes. While the additional historical data that we require can often be obtained, the fault probabilities for individual components are often not widely available.
5. We do not take different operating modes into account and thus need to assume there is only a normal working mode.

Overall we have presented a new method to compute novel observables using algorithm SynthesizeObs. For this we use the well-known tuple (SD, COMPS, OBS) along with historical data in the form of traces for the computation of critical pairs. The critical pairs

provide us with the information which components within a system are non-diagnosable. We then use algorithm SynthesizeObs to compute optimal novel observables that include synthesized normal operating ranges which can thus be used for anomaly detection and fault diagnosis. If information about individual fault probability is given, then we also compute the entropy-maximal novel observable, i.e. that observable whose addition in the system would gather the most information.

We hope that our method aids practitioners in improving the observability particularly of learned system descriptions in industrial use-cases.

References

1 Mehdi Bayoudh and Louise Travé-Massuyès. Diagnosability analysis of hybrid systems cast in a discrete-event framework. *Discrete Event Dynamic Systems*, 24(3):309–338, 2014. doi:10.1007/S10626-012-0153-Z.

2 Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, Marco Gario, Stefano Tonetta, and Viktoria Vozarova. Diagnosability of fair transition systems. *Artificial Intelligence*, 309:103725, 2022. doi:10.1016/J.ARTINT.2022.103725.

3 Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, and Xavier Olive. Symbolic synthesis of observability requirements for diagnosability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 712–718, 2012. doi:10.1609/AAAI.V26I1.8225.

4 Alexander Bochman. *A logical theory of causality*. MIT Press, 2021.

5 Abderraouf Boussif and Mohamed Ghazel. Diagnosability analysis of intermittent faults in discrete event systems using a twin-plant structure. *International Journal of Control, Automation and Systems*, 18(3):682–695, 2020.

6 Marco Bozzano, Alessandro Cimatti, and Stefano Tonetta. Testing diagnosability of fair discrete-event systems. In *Proc. International Workshop on Principles of Diagnosis (DX-19)*, 2019.

7 Maria Paola Cabasino, Alessandro Giua, Stéphane Lafortune, and Carla Seatzu. A new approach for diagnosability analysis of petri nets using verifier nets. *IEEE Transactions on Automatic Control*, 57(12):3104–3117, 2012. doi:10.1109/TAC.2012.2200372.

8 Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. Enforcing resilience in cyber-physical systems via equilibrium verification at runtime. *ACM Transactions on Autonomous and Adaptive Systems*, 2023.

9 Cody James Christopher and Alban Grastien. Critical observations in model-based diagnosis. *Artificial Intelligence*, page 104116, 2024. doi:10.1016/J.ARTINT.2024.104116.

10 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

11 Marie-Odile Cordier, Louise Travé-Massuyes, Xavier Pucel, et al. Comparing diagnosability in continuous and discrete-event systems. In *Proceedings of the 17th international workshop on principles of diagnosis (DX-06)*, pages 55–60, 2006.

12 Johan De Kleer and Brian C Williams. Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

13 Maria D Di Benedetto, Stefano Di Gennaro, and Alessandro D'Innocenzo. Verification of hybrid automata diagnosability by abstraction. *IEEE transactions on automatic control*, 56(9):2050–2061, 2011. doi:10.1109/TAC.2011.2105738.

14 Alexander Diedrich, Patrick Deutschmann, and Caroline Junker. Servicenavigator - a bayesian assistance system for diagnosing industrial production systems. In *2022 5th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2022.

15 Alexander Diedrich, Alexander Maier, and Oliver Niggemann. Model-based diagnosis of hybrid systems using satisfiability modulo theory. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19), 2019.

16 Alexander Diedrich, Lukas Moddemann, and Oliver Niggemann. Learning system descriptions for cyber-physical systems. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

17 Alexander Diedrich and Oliver Niggemann. Diagnosing systems through approximated information. *13th Annual Conference of the Prognostics and Health Management Society*, 2021.

18 Alexander Diedrich and Oliver Niggemann. On residual-based diagnosis of physical systems. *Engineering Applications of Artificial Intelligence*, 109:104636, 2022. doi:10.1016/j.engappai.2021.104636.

19 Oumar Diene, Marcos V Moreira, Eduardo A Silva, Victor R Alvarez, and Claudionor F Nascimento. Diagnosability of hybrid systems. *IEEE Transactions on Control Systems Technology*, 27(1):386–393, 2017.

20 Barry Dowdeswell, Roopak Sinha, and Stephen G MacDonell. Diagnosable-by-design model-driven development for iec 61499 industrial cyber-physical systems. In *IECON 2020 the 46th annual conference of the IEEE industrial electronics society*, pages 2183–2188. IEEE, 2020. doi:10.1109/IECON43393.2020.9254620.

21 Jonas Ehrhardt, Malte Ramonat, Rene Heesch, Kaja Balzereit, Alexander Diedrich, and Oliver Niggemann. An ai benchmark for diagnosis, reconfiguration & planning. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2012.

22 Teresa Escobet, Anibal Bregon, Belarmino Pulido, and Vicenç Puig. *Fault Diagnosis of Dynamic Systems*. Springer, 2019.

23 Alexander Feldman, Gregory Provan, and Arjan Van Gemund. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research*, 38:371–413, 2010. doi:10.1613/JAIR.3025.

24 GK Fourlas, Kostas J Kyriakopoulos, and NJ Krikelis. Diagnosability of hybrid systems. In *Proceedings of the 10th IEEE mediterranean conference on control and automation*, 2002.

25 Zhiwei Gao, Carlo Cecati, and Steven X Ding. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3757–3767, 2015. doi:10.1109/TIE.2015.2417501.

26 Alban Grastien and Marina Zanella. Diagnosability of discrete faults with uncertain observations. *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems*, pages 253–278, 2018.

27 Daniel Jung. Structural methods for distributed fault diagnosis of large-scale systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 2690–2695. IEEE, 2020. doi:10.1109/CDC42340.2020.9303744.

28 Samuel Kolb, Stefano Teso, Andrea Passerini, and Luc De Raedt. Learning smt (lra) constraints using smt solvers. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2018, pages 2333–2340. ijcai. org, 2018. doi:10.24963/IJCAI.2018/323.

29 Mattias Krysander and Mattias Nyberg. Fault diagnosis utilizing structural analysis. *CCSSE, Norrköping, Sweden*, 2002.

30 Stephane Lafortune and Feng Lin. From diagnosability to opacity: A brief history of diagnosability or lack thereof. *IFAC-PapersOnLine*, 50(1):3022–3027, 2017.

31 Gianfranco Lamperti, Stefano Trerotola, Marina Zanella, and Xiangfu Zhao. Sequence-oriented diagnosis of discrete-event systems. *Journal of Artificial Intelligence Research*, 78:69–141, 2023. doi:10.1613/JAIR.1.14630.

32 Ruben Leal, Jose Aguilar, Louise Travé-Massuyès, Edgar Camargo, and Addison Ríos. A genetic algorithm approach for diagnosability analysis. *International Journal of Engineering Development and Research*, 2(4):3786–3799, 2014.

33 Alexander Maier. Online passive learning of timed automata for cyber-physical production systems. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, pages 60–66. IEEE, 2014. doi:10.1109/INDIN.2014.6945484.

34 Mohammadreza Mehrabian, Mohammad Khayatian, Aviral Shrivastava, Patricia Derler, and Hugo Andrade. A run-time verification method with consideration of uncertainties for cyber-physical systems. *Microprocessors and Microsystems*, page 104890, 2023. doi:10.1016/J.MICPRO.2023.104890.

35 Amit Metodi, Roni Stern, Meir Kalech, and Michael Codish. A novel sat-based approach to model based diagnosis. *Journal of Artificial Intelligence Research*, 51:377–411, 2014. doi:10.1613/JAIR.4503.

36 Lukas Moddemann, Henrik Sebastian Steude, Alexander Diedrich, and Oliver Niggemann. Discret2di - deep learning based discretization for model-based diagnosis. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

37 Fatiha Nejjari, Ramon Pérez, Teresa Escobet, and Louise Travé-Massuyès. Fault diagnosability utilizing quasi-static and structural modelling. *Mathematical and computer modelling*, 45(5-6):606–616, 2007. doi:10.1016/J.MCM.2006.06.008.

38 Ingo Pill and Franz Wotawa. Fault detection and localization using modelica and abductive reasoning. *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems*, pages 45–72, 2018.

39 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

40 Meera Sampath, Stéphane Lafortune, and Demosthenis Teneketzis. Active diagnosis of discrete-event systems. *IEEE transactions on automatic control*, 43(7):908–929, 1998. doi:10.1109/9.701089.

41 Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995. doi:10.1109/9.412626.

42 Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. doi:10.1002/J.1538-7305.1948.TB01338.X.

43 Roni Stern, Meir Kalech, and Orel Elimelech. Hierarchical diagnosis in strong fault models. In *Twenty Fifth International Workshop on Principles of Diagnosis*, 2014.

44 Jinya Su and Wen-Hua Chen. Model-based fault diagnosis system verification using reachability analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(4):742–751, 2017.

45 L Travé-Massuyes, T Escobet, and R Milne. Model-based diagnosability and sensor placement. *12th Intl. Work. on Principles of Diagnosis*, 2001.

46 Louise Travé-Massuyès and Teresa Escobet. Bridge: Matching model-based diagnosis from fdi and dx perspectives. In *Fault Diagnosis of Dynamic Systems*, pages 153–175. Springer, 2019.

47 Louise Travé-Massuyes, Teresa Escobet, and Xavier Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 36(6):1146–1160, 2006. doi:10.1109/TSMCA.2006.878984.

48 Ángel Jesús Varela-Vaca, David G Rosado, Luis E Sánchez, María Teresa Gómez-López, Rafael M Gasca, and Eduardo Fernández-Medina. Carmen: A framework for the verification and diagnosis of the specification of security requirements in cyber-physical systems. *Computers in Industry*, 132:103524, 2021. doi:10.1016/J.COMPIND.2021.103524.

49 Nathalie Verdière, Carine Jauberthie, and Louise Travé-Massuyès. Functional diagnosability and detectability of nonlinear models based on analytical redundancy relations. *Journal of Process Control*, 35:1–10, 2015.

50 Amaury Vignolles, Elodie Chanthery, and Pauline Ribot. An overview on diagnosability and prognosability for system monitoring. In *European conference of the Prognostics and Health Management Society (PHM Europe)*, 2020.

51 Su Xingyu, Marina Zanella, Alban Grastien, et al. Diagnosability of discrete-event systems with uncertain observations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 1265–1271. AAAI Press/International Joint Conferences on Artificial Intelligence, 2016.

Leveraging Answer Set Programming for Continuous Monitoring, Fault Detection, and Explanation of Automated and Autonomous Driving Systems

Lorenz Klampfl[1]
CD Laboratory QAMCAS, Institute of Software Technology, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Technology, Graz University of Technology, Austria

Abstract

Recent advancements in automated and autonomous driving systems have facilitated their integration into modern vehicles, enabling them to accurately perceive their surroundings and support or even fully undertake complex driving tasks. Given the complexity and unpredictable nature of driving environments and traffic situations, ensuring the correct behavior of such systems is essential to prevent hazardous situations, increase user acceptance, and avoid human harm. However, the increased complexity of these systems and the extensive search space of possible scenarios introduce significant challenges to testing and real-time fault management. Hence, besides rigorous testing during the development phase, there is a need for additional validation and verification during operation. This paper proposes utilizing Answer Set Programming (ASP), a form of declarative programming, for continuous real-time monitoring, fault detection, and explanation to ensure the correct functioning of automated and autonomous driving systems. Our approach aims to enhance the reliability and safety of such systems by detecting violations and providing explanations that can support fault-adaptive control or mitigation strategies. We demonstrate the effectiveness of our methodology across diverse scenarios executed within a simulation environment, discuss the main challenges encountered, and outline future research directions.

2012 ACM Subject Classification General and reference → Validation

Keywords and phrases Autonomous Driving, Answer Set Programming, Continuous Monitoring

Digital Object Identifier 10.4230/OASIcs.DX.2024.10

Funding The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, and the Christian Doppler Research Association is gratefully acknowledged.

1 Introduction

In recent years, newly developed algorithms and software components utilizing artificial intelligence (AI) have become central components integrated into modern vehicles. Smart sensors reliably perceiving the vicinity of autonomous vehicles, as well as advanced algorithms for path planning and handling complex driving situations, successfully demonstrate how Advanced Driver-Assistance Systems (ADAS) and Autonomous Driving (AD) technologies can contribute to the vision of safe, efficient, and comfortable transportation. However, guaranteeing safety and robustness under all circumstances during driving remains an open challenge. In both engineering and research, the necessity for comprehensive testing and validation to comply with established standards and regulations, such as UN Regulation No. 157 [3], and

[1] Corresponding Author

© Lorenz Klampfl and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0

to ensure the reliability and trustworthiness of these systems has been widely recognized, e.g., Wotawa and colleagues [20], Wotawa [17], and Koopman et al. [11]. It is essential to consider that autonomous vehicles operate within domains characterized by unpredictability and uncertainty, necessitating robust safety measures and testing methodologies. It is evident that traditional mileage validation is impractical, as expressed by Kalra and Paddock [7], stating that autonomous vehicles must operate 275 million miles for verification. As a result, new methodologies have emerged over the last few years, emphasizing the advantages of using virtual validation approaches and simulation frameworks to test ADAS/AD effectively, e.g., Klampfl et al. [8], Schuldt et al. [13] or Klück and colleagues [10]. Nevertheless, despite the considerable advancements virtual validation methods have introduced compared to on-road testing, these approaches also come with limitations. Considering the range of parameters involved, e.g., environmental conditions, traffic signs, or road characteristics, alongside the complex dynamics between other road users and the autonomous vehicle, verifying the correct behavior for all scenarios within reasonable time seems unfeasible. Furthermore, there may be unknown factors or a combination of factors that could lead to ADAS or AD system malfunctioning. Unfortunately, numerous incidents in recent years have underscored the complexities and challenges in testing and verifying autonomous driving systems. This highlights the need for innovative solutions that ensure the continuous safe operation of autonomous vehicles and their correct responses to unforeseen or unknown events.

Our Approach

In this paper, we aim to enhance the safety and reliability of ADAS/AD by introducing an additional layer of quality assurance that can be deployed during vehicle operation and is responsible for continuously monitoring and validating the behavior of autonomous vehicles during operation, complementary to strict testing during the development phase of automated and autonomous driving systems. Figure 1 depicts a high-level overview of our proposed approach, combining vehicle sensors from an autonomous car or simulation data from a simulation environment and a continuous validation setup (CVS) comprising formalized logic-based expectations for vehicle behavior and solving mechanisms for monitoring, identifying, and explaining potential ADAS/AD misbehavior. Through repeated interactions between the autonomous vehicle or a simulation environment and the CVS, we aim to find discrepancies between expected and actual behavior. It is worth noting that the general principles of the CVS can be similarly implemented across different layers of abstraction, e.g., raising and diagnosing sensor faults or recognizing physical impossibilities of objects detected by the perception system.

Our Contributions

Within this paper, we contribute to the topic of safe and robust behaviors of deployed automated and autonomous driving systems based on the actions they perform. Specifically, we investigate four different prototypical Automated Lane Keeping Systems (ALKS). In addition, we assume that vehicle sensors work as expected and that the perception system correctly detects and classifies all objects in the vicinity of the autonomous vehicle. This assumption is ensured by generating virtual scenarios executed within a simulation environment, i.e., Esmini[2], where ground truth data about the vehicle's state, as well as positions and states of other road users and objects, is available. Our primary contribution includes the development

[2] `https://github.com/esmini/esmini`

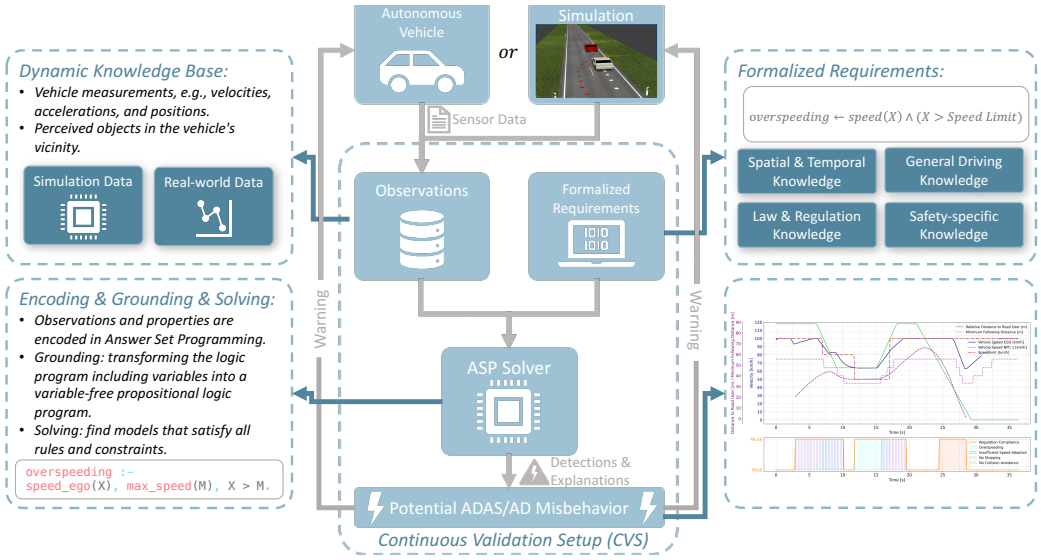


Figure 1 High-level schematic showing the general concept of our approach and how the framework's different components interact.

of a model incorporating formalized logic-based properties that an ALKS has to comply with to ensure correct behavior. In more detail, we provide a model covering parts of the requirements stated in regulation UN R157 [3] for Automated Lane Keeping Systems. We leverage Answer Set Programming (ASP) [2] and the ASP solver Clingo [3] [4], which is using an extended version of Prolog [1] as input language, to encode the regulations an ALKS has to comply with. Using a declarative programming approach like ASP, combined with advanced solvers like Clingo, allows us to efficiently monitor, detect faults, explain possible misbehavior, and extend our model with additional or new rules without implementing an entirely new model. Furthermore, we demonstrate the effectiveness of our model by developing a framework and corresponding interfaces to integrate our requirement compliance model into a simulation framework, enabling continuous interaction between our model and the simulation.

Outline

The structure of our paper is as follows: First, we give a brief introduction to ASP, followed by a description of our proposed model, detailing its design, implementation, and evaluation. Finally, we discuss the implications of our findings and highlight the potential of ASP to enhance the safety and reliability of Advanced Driver Assistance Systems (ADAS) and autonomous driving (AD).

2 Background

The following paragraphs outline the theoretical and technical foundation for our research and experimental evaluation, including a brief overview of ASP, our simulation framework, and continuous validation setup.

[3] `https://potassco.org`

2.1 Answer Set Programming (ASP)

Answer Set Programming is a programming approach that emphasizes specifying the problem in a declarative way rather than in an imperative. ASP originates from several fields, including deductive databases, logic programming, and knowledge representation and reasoning. It is applied in various domains, such as planning, configuration, decision support, model checking, robotics, and autonomous driving. In recent years, ASP has gained particular attention in the automotive domain. For instance, Suchan and colleagues introduced a modular framework for visual sensemaking using ASP and a qualitative model for tracking objects in traffic, including occlusion handling, as detailed in their works [15] and [16]. Similarly, Wotawa and Klampfl utilized logic models and ASP to identify critical situations in autonomous driving during operation [9]. Additionally, Gilpin proposed methods for monitoring autonomous driving systems and comparing them with driving knowledge, regulatory knowledge, and safety requirements [5].

ASP is based on the stable model (or answer set) semantics of logic programming. Given a program P, its answer sets can be understood as the intended models of P, each representing a distinct scenario or world described by P. An ASP program is a collection of rules of the form:

$$a_0 \leftarrow a_1, ..., a_m, not\ a_{m+1}, ..., not\ a_n. \tag{1}$$

where $(a_0, ..., a_n)$ are atoms and *not* represents negation as failure. The semantics of ASP facilitate the representation of default assumptions and exceptions, which is crucial for modeling the dynamic and uncertain environments encountered by autonomous vehicles.

Let us consider the following example of approaching a traffic light. We can express a rule stating that we are only allowed to drive when the traffic light is green as follows:

$$colorDrive(green). \tag{2}$$

$$stop \leftarrow trafficLightColor(X), not\ colorDrive(X). \tag{3}$$

$$noStoppingTrafficLight \leftarrow stop, not\ stoppingInitiated. \tag{4}$$

Here, (2) encodes the fact that we are only allowed to drive if the traffic light color is green. (3) is the formalization of the rule that stopping is required using ASP syntax with *not* representing negation as failure, i.e., *not* $colorDrive(X)$ is $TRUE$ if all attempts to prove $colorDrive(X)$ fail. Therefore, for all other traffic light colors X, we would derive the atom *stop* indicating that our autonomous vehicle needs to stop at the present traffic light. Furthermore, with rule (4), where *not stoppingInitiated* represents, for instance, an observation delivered by a sensor of an autonomous vehicle, we can derive a faulty behavior of the system under test and its corresponding explanation, i.e., the autonomous vehicle has not stopped at a traffic light although the traffic light color was not green.

It is worth mentioning that ASP has been already used to implement Model-based Diagnosis (see [18, 19]). In the context of this paper, we utilize MBD for implementing fault detection based on models, i.e., the formalized regulations, and reasoning.

Solving Procedure

The solving procedure for ASP can be divided into two main phases: grounding and solving.

1. **Grounding:** The grounding phase translates a high-level ASP program, which may contain variables and complex expressions, into a variable-free propositional logic program. This process involves generating all possible instances of the rules by substituting variables

with constants from the domain. The challenge in grounding is to manage the exponential growth in the size of the program, which can lead to what is known as the "grounding bottleneck." To mitigate this, grounding tools included in solvers like Clingo apply intelligent techniques to reduce the size of the grounded program, such as eliminating irrelevant rules and facts.

2. **Solving:** Once the program is grounded, the solving phase involves searching for answer sets of the grounded program that satisfy all rules and constraints included in the logic program. Therefore, in our case, our answer sets represent potential misbehaviors of the system under test and its corresponding explanations.

For a more detailed explanation of the grounding and solving procedure, we refer the interested reader to the respective literature, e.g., [4].

2.2 Simulation Framework and Continuous Validation Setup

Figure 1 illustrates a high-level overview of our approach's concept, including the different artifacts implemented and their interactions. Our methodology comprises two main components: the simulation framework or autonomous vehicle and the continuous validation setup (CVS). The simulation framework, representing the autonomous vehicle in our use cases, is responsible for providing sensor data. This data is used to generate a dynamic knowledge base during runtime, which includes all necessary information to determine whether the ALKS under test complies with all implemented regulations and expected behaviors. Measurements of particular importance include accelerations and the positions of all objects in the simulation environment. It is important to note that during our experiments, we only used data gathered from simulation to evaluate our approach. This ensures that all additional road users are correctly identified, particularly regarding their positions, allowing us to evaluate the behavior of the four investigated ALKSs without interference from detection and classification algorithms. As already mentioned, we decided to use Esmini as our simulator. Esmini is an environment simulator for replaying ASAM OpenSCENARIO [4] files, which has become a widely adopted standard for virtual development, testing, and validation in the autonomous driving domain. We utilize Esmini to load, execute, and collect simulation data during scenario execution. Detailed descriptions of the scenarios used in our experiments, as well as the integrated ALKS controllers, are provided in subsequent sections. The CVS comprises the dynamic knowledge base, the formalized requirements (see Table 1), the interface to our ASP solver Clingo, and a result post-processing pipeline. At each simulation time step, i.e., every 0.1 seconds, the CVS retrieves simulation data, encodes the provided observations to ASP syntax, grounds our logic program, and solves it by finding all answer sets that satisfy our rules and constraints. Since simulation data cannot be directly utilized by the CVS, an encoding approach similar to that proposed by Jahaj et al. [6] has been implemented. Specifically, within each time step, a set of adaptors is used to convert the incoming sensor data into ASP facts. For example, an adaptor may take the input from the speed sensor and convert the received data into the fact `speed(130)`. Therefore, for each adaptor, we have a function that implements the conversion from the sensor domains to ASP facts.

[4] `https://www.asam.net/`

3 UN R157 Requirement Formalization

Table 1 showcases the requirements we implemented to test our methodology. As seen in the table, only a subset of requirements was included, namely those dealing with system activation, general traffic rules, and the dynamic driving task. Table 1 depicts from which chapter the requirement was extracted, a short description of the requirement, as well as a general formalization of the requirement demonstrating how we implemented it within our logic model. It is worth noting that all requirements were implemented within one model. With this, we ensure that at every time step, all requirements are checked and possible faults are detected and explained appropriately. As an example, let us consider the first requirement in the table, i.e., *System Activation*. Within regulation UNR157, it is stated that an ALKS system can only be activated under certain conditions, i.e., the automated or autonomous vehicle is driving on a specific road characterized by a physical separation dividing the traffic moving in opposite directions and preventing traffic from cutting across the street. Furthermore, road types where pedestrians and cyclists are prohibited are a prerequisite for successfully activating the ALKS. Concerning vehicle speed, the ALKS is only allowed to be activated up to operational speeds of 60 km/h. However, if the ALKS includes the functionality of performing Minimum Risk Maneuvers (MRM), the maximum operational speed increases to 130 km/h. Those requirements are implemented with three dedicated rules as stated in Table 1. Each rule would infer *prohAct* in case of a prohibited activation of the ALKS system. The first rule evaluates if the road type we are actually driving on is suitable for activating the ALKS, which is encoded with the fact *alksRoad*, and if the ALKS system is activated, i.e., *alksActive*. In case *alksActive* is true, and *alksRoad* is false, the term *not alksRoad* will evaluate to true, resulting in deriving the fact *prohAct*. Similarly, the second and third rule evaluate if *alksActive* is true and check if the vehicle speed is within the allowed range. Furthermore, the second rule includes the term *not alksMRM*, which evaluates to true in case the ALSK system is not capable of performing a Minimum Risk Maneuver. All other requirements are implemented similarly. For a detailed description of the UN R157 requirements, we refer the interested reader to [3].

4 Unit Testing Evaluation

In order to demonstrate the correct implementation and outcomes of our CVS, we conducted two separate evaluation approaches: a unit testing approach as well as an experimental evaluation using a dedicated simulation framework (see Chapter 5). We implemented a total of 41 unit tests, covering all requirements outlined in Table 1. To leverage Clingo's dedicated interfaces to Python, we used Python[5], version 3.12.2, and the PyTest[6] framework, version 7.4.3, to write and execute our unit tests. PyTest is a simple yet powerful framework that simplifies the process of writing and running tests by providing a clean and readable syntax. It also supports parameterized testing, allowing us to test, for instance, multiple vehicle speed values within a single unit test.

Table 2 presents three example unit tests, describing their parameters and values. Each unit test comprises different parameters and their respective values, meaning that each test evaluates the model based on specific parameters and their combinations. For the unit testing framework, we implemented dedicated Python classes and functions to encode these

[5] `https://www.python.org`
[6] `https://docs.pytest.org/en/8.2.x/index.html`

Table 1 Overview depicting the implemented requirements included in UN Regulation No. 157 [3].

UN R157 Requirements		
Reference	**Name**	**Short Description**
Formalization		
Intro 5.2.3.1.	System Activation	The system can be activated only under certain conditions, e.g., suitable roads or speeds up to 60 km/h. If the ALKS is equipped with a Minimum Risk Maneuver(MRM) functionality, the maximum speed limit for activation increases to 130 km/h.
$prohAct \leftarrow alksActive \wedge not\ alksRoad.$ $prohAct \leftarrow alksActive \wedge speed(X) \wedge (X > 60) \wedge not\ alksMRM.$ $prohAct \leftarrow alksActive \wedge speed(X) \wedge (X > 130).$		
5.1.2	Traffic Rules	The system shall comply with traffic rules, for instance, obey speed limits. Additional regulations, such as the requirement to stop at stop signs or traffic lights, are encoded similarly.
$overspeeding \leftarrow speed(X) \wedge speedSign(Limit) \wedge (X \geq Limit).$ $overspeeding \leftarrow maxSpeed(Country, Streettype, Limit) \wedge speed(X) \wedge (X \geq Limit).$		
5.2.1	Dynamic Driving Task	The system shall ensure the vehicle's lateral stability and position within its lane, avoiding unintentional lane departures. Longitudinal rules are implemented similarly considering accelerations. $Thres$ defines at which distance to a lane marking it is considered as crossing.
$unintLaneChane \leftarrow distLeft(DistLeft) \wedge (DistLeft < Thres) \wedge not\ laneChangeInitiated.$ $unintLaneChane \leftarrow distRight(DistRight) \wedge (DistRight < Thres) \wedge not\ laneChangeInitiated.$		
5.2.2	Dynamic Driving Task	The system shall appropriately adjust the speed and the lateral position within its lane in case another vehicle is detected in an adjacent lane. The system shall not accelerate and ensure that the vehicle is not too close to the lane boundaries.
$noAdjustment \leftarrow nextToDrivingLane(Vehicle) \wedge not\ keepingSpeed.$ $noAdjustment \leftarrow nextToDrivingLane(Vehicle) \wedge not\ centricPosition.$		
5.2.3	Dynamic Driving Task	The system shall control the vehicle's speed and adapt it when certain conditions are present, e.g., environmental conditions or a vehicle in front. $sufficientSpeedAdaption$ is considered $True$ when the vehicle speed is lower than the allowed speed concerning properties like environmental conditions (speed should be reduced appropriately) as well as if the distance to the detected vehicle in front is greater than the minimum following distance to the vehicle based on the table depicted in [3], paragraph 5.2.3.3.
$noSpeedAdjustment \leftarrow envCondition(Env) \wedge not\ sufficientSpeedAdaption(Env).$ $noSpeedAdjustment \leftarrow roadUserOnLane(Vehicle) \wedge not\ sufficientSpeedAdaption(Vehicle).$		
5.2.4	Dynamic Driving Task	The system shall bring the vehicle to a complete stop behind a stationary object. $collisionAvoidable$ is considered $True$ if the relative distance to the object is greater than the minimum following distance.
$noStoppingManeuver \leftarrow objectOnLane(Obj) \wedge collisionAvoidable \wedge not\ stoppingInitiated.$		
5.2.5	Dynamic Driving Task	The system shall detect the risk of a collision and shall perform appropriate maneuvers, e.g., lane change or emergency maneuver. $collisionAvoidable$ is considered $True$ if the relative distance to the object is greater than the minimum following distance.
$noCollisionAvoidance \leftarrow$ $objectOnLane(Obj) \wedge not\ collisionAvoidable \wedge not\ collisionAvoidanceInitiated.$		

Table 2 Overview of some of the executed unit tests based on the requirements stated within Table 1.

UN R157 Requirements - Unit Tests		
Reference	**Name**	**Description**
Inputs		
Intro 5.2.3.1.	System Activation	Three dedicated unit tests were implemented covering the testing of a prohibited ALKS activation for unsuitable road types as well as for a prohibited activation at certain speeds dependent on the availability of Minimum Risk Maneuver (MRM) functionality.
$RoadTypes = [0, ruralRoad, city, motorway]$ $VehicleSpeeds = [0, 50, 59.999, 60, 60.001, 100, 129.999, 130, 130.001, 150, 200]$ $MRMFunctionality = [True, False]$		
5.1.2	Traffic Rules	Three unit tests were performed to check if overspeeding is correctly identified. This includes a differentiation between the situation where a speed sign is present and not as well as signs that restrict the speed to a lower or higher maximum allowed speed.
$RoadTypes = [motorway]$ $VehicleSpeeds =$ $[0, 95, 99.999, 100, 100.001, 120, 125, 129.999, 130, 130.001, 135, 139.999, 140, 140.001, 145, 200]$ $SpeedSignLimit = [notavailable, 100, 140]$		
5.2.3	Dynamic Driving Task	The system shall control and adapt the speed of the vehicle when certain conditions are present, e.g., environmental conditions or a vehicle in front. A dedicated unit test was implemented for both. In case of environmental conditions like rain, we assume that a sufficient speed reduction is achieved when the vehicle speed is equal to or lower than 75% of the maximum allowed speed, i.e., for motorways in Austria where the speed limit is 130km/h, this would result in 97.5 km/h. For a detected road user, sufficient speed reduction is achieved when the vehicle obeys the minimum following distance based on the table depicted in [3], paragraph 5.2.3.3.
$RoadTypes = [motorway], EnvironmentCondition = [rain]$ $VehicleSpeeds =$ $[0, 50, 69.999, 70, 70.001, 79.999, 80, 80.001, 97.499, 97.500, 97.501, 100, 129.999, 130, 130.001, 150, 200]$ $DistanceToRoadUser = [notavailable, 40]$		

parameters and values into the appropriate ASP syntax for passing them to the Clingo solver and for post-processing the results. As shown in the provided examples, we included boundary values within the allowable ranges to identify potential errors and defects in our model. This approach ensures that any issues arising from incorrect assumptions are detected.

All 41 unit tests were successfully passed, providing a solid foundation for further evaluation within a simulation environment that includes an actual ALKS controller and test scenarios.

5 Experimental Evaluation

Building on the results obtained from unit testing, we conducted an extensive experimental evaluation to demonstrate the applicability of our proposed methodology within a simulation environment, including a vehicle equipped with a prototypical ALKS. As previously mentioned, we established a simulation framework using Esmini and developed dedicated interfaces to integrate our logic model. These interfaces facilitate continuous communication between the simulation environment and the CVS, ensuring that sensor data is transmitted seamlessly.

The continuous validation setup (CVS) processes the received simulation data every 0.1 seconds. This involves encoding the data into the appropriate syntax and determining whether potential misbehavior is occurring. To test our model on actual ALKS controllers, we integrated four different controllers available within Esmini. Each controller was subjected to 977 automatically executed and evaluated scenarios, totaling in 3,908 test scenarios.

The following sections provide an overview of several example scenarios, detailing the maneuvers involved, the integrated ALKS controllers, and the evaluation outcomes. This thorough experimental evaluation underscores the practical applicability of our methodology, showcasing its effectiveness in ensuring safe and reliable behavior in automated and autonomous driving systems.

5.1 ALKS Controllers

Advanced Lane Keeping Systems are designed to control the lateral and longitudinal movement of a vehicle for extended periods without human intervention. Since obtaining sophisticated controllers for automated or autonomous driving from companies is challenging due to their proprietary nature, it is fortunate that Esmini includes several pre-built controllers for vehicle control within their simulations framework. According to Esmini's user guide, various controllers are available, such as an interactive control using a simple vehicle model that can be operated via keyboard inputs and more advanced controllers based on a ghost concept, where a ghost-twin performs planned events in a few seconds ahead, and the vehicle under control mimics its behavior. In addition, Esmini provides a pre-built ALKS controller named *ALKS_R157SM*, inspired by the safety models described in UN Regulation No. 157 [3]. This controller offers four different modes (safety models), though it is important to note that these implementations are preliminary and experimental and not officially commissioned ALKS controllers. The four safety models for the controller are described as follows:

1. **Regulation:** This model includes the characteristics of the expected system performance, which is stated within paragraph 5.2.5.2 in [3].
2. **ReferenceDriver:** Within Annex 4 - Appendix 3 of [3], a reference model of a "typical" human driver is specified, which is represented by this safety model.
3. **RSS:** Responsibility-Sensitive Safety: This mode includes the proposed performance model described by Shalev-Shwartz and colleagues [14].
4. **FSM:** Fuzzy Safety Model was introduced and described by Konstantinos et al. [12].

Since the objective of this paper is not to describe the inner workings of each safety model and the integrated ALKS controller, we refer interested readers to the respective literature. However, all these controllers share the capability of maintaining longitudinal and lateral stability and reacting appropriately to other road users. However, none of the models has advanced steering and emergency maneuver capabilities. In the next section, an overview of four example use cases and a description of the scenario properties will be given.

5.2 Test Scenarios

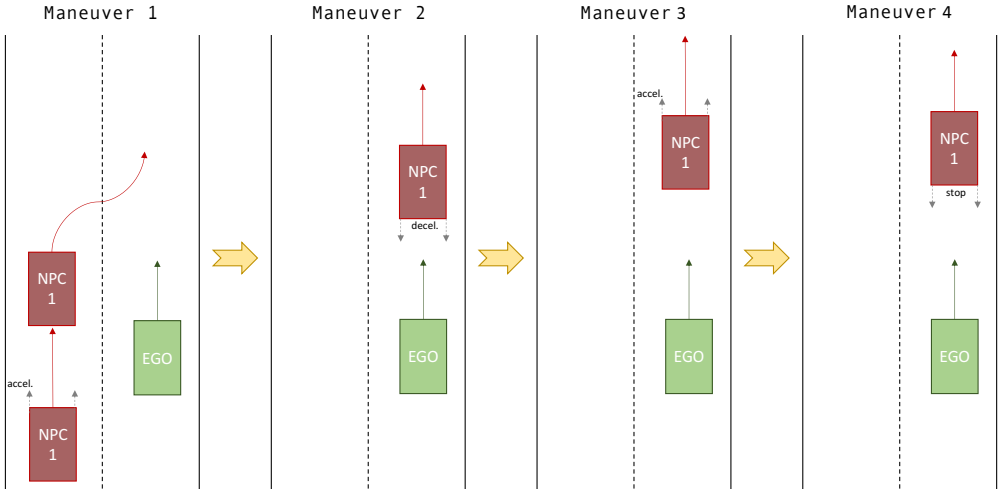
All of the executed and evaluated test scenarios are based on the ASAM OpenSCENARIO and ASAM OpenDRIVE [7] standard. OpenSCENARIO is a widely adopted standard for defining the dynamic part of complex traffic scenarios in a detailed and structured manner. Furthermore, it provides an extensible framework for specifying the behavior of all entities in

[7] https://www.asam.net/

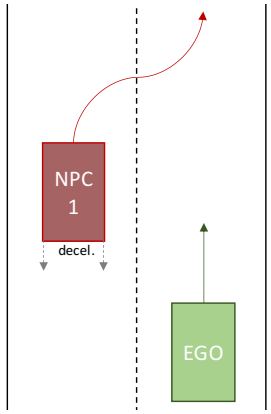
the simulation, e.g., additional road users, pedestrians, and environmental factors. This allows us the precise modeling of various driving situations, enabling thorough testing and validation of automated driving systems under a variety of conditions. In contrast, OpenDRIVE is used to define the static properties of the scenario, e.g., road networks, traffic signs, or road geometry, ensuring that the environment is accurately represented. Both standards interact closely with each other, as OpenSCENARIO uses the road network data to position and move all entities within the simulation. Utilizing both of these well-established standards ensures that our test scenarios are not only realistic and reproducible but also compatible with industry practices. The integration of these standards enhances the reliability and relevance of our experimental evaluation, providing a comprehensive framework for testing and validating the behavior of the integrated ALKS with our proposed methodology for continuous monitoring, fault detection, and explanation using ASP.

Although we generated a total of 977 scenarios per ALKS controller for evaluating our approach, in this paper, we want to highlight four representative test scenarios and the results we obtained from them. Figure 2 illustrates the maneuvers performed within each scenario when executed within Esmini. Before detailing the sample scenarios, it is important to note that *NPC1* represents an additional road user within the scenario whose trajectories and actions are predefined within the OpenSCENARIO files. *EGO* represents the automated vehicle equipped with the previously explained ALKS controller that should accurately react to maneuvers performed by *NPC1*. Below, we provide a brief description of each scenario along with the expected behavior of the automated vehicle.

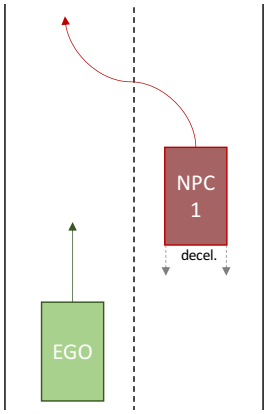
1. **ALKS R157 Test Scenario (Figure 2a)**
   Scenario Description: This scenario is already included within Esmini, however, we slightly adapted the starting positions and vehicle speeds as well as inserted speed limit signs. As depicted, the scenario consists of four separate maneuvers executed consecutively. The *EGO* vehicle is driving in the right lane and is overtaken by *NPC1*, which accelerates and performs a cut-in maneuver in front of the automated vehicle. After the successful cut-in, *NPC1* decelerates in front of the *EGO* vehicle. In the last two maneuvers of this scenario *NPC1* accelerates before coming to a complete stop.
   Expected EGO Behavior: After the first cut-in, a safe following distance should be re-established. During the deceleration phase in maneuver two, the *EGO* should react accordingly and also decelerate. For the stopping maneuver, *EGO* should detect the obstacle in front and come to a stop as well since the controllers are not equipped with lane change capabilities. Within the complete scenario, the *EGO* should avoid speed limit violations.
2. **Cut-In (Right) & Deceleration (Figure 2b)**
   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* starts in the left adjacent lane with a longitudinal offset positioned ahead of the *EGO*. *NPC1* then performs a cut-in maneuver into the *EGO*'s lane and begins decelerating.
   Expected EGO Behavior: Initially, the *EGO* should continue driving without considering *NPC1* since it is in an adjacent lane. Once *NPC1* cuts in front of the *EGO*, the *EGO* should detect *NPC1* as a target object and maintain a safe following distance behind it.
3. **Cut-In (Left) & Deceleration (Figure 2c)**
   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* starts in the right adjacent lane with a longitudinal offset positioned ahead of the *EGO*. *NPC1* then performs a cut-in maneuver into the *EGO*'s lane and begins decelerating.



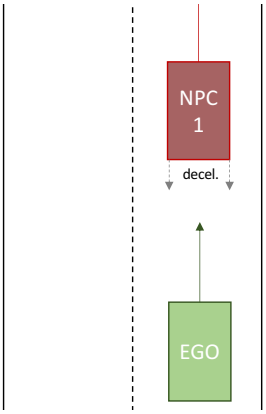
(a) ALKS R157 Test Scenario.



(b) Cut-In (Right) & Deceleration.



(c) Cut-In (Left) & Deceleration.



(d) Same Lane & Deceleration.

Figure 2 Representative scenarios for testing our approach for continuous monitoring, fault detection, and explanation.

Expected EGO Behavior: Initially, the *EGO* should not recognize *NPC1* as a target object since it is in an adjacent lane. Once *NPC1* cuts in front of the *EGO*, the *EGO* should detect *NPC1* as a target object and maintain a safe following distance to avoid a collision.

4. **Same Lane & Deceleration (Figure 2d)**

   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* is driving at the same speed in the same lane, positioned ahead of the *EGO*. *NPC1* then begins to decelerate.

   Expected EGO Behavior: The *EGO* recognizes *NPC1* as a target object and follows it at a constant speed. Once *NPC1* starts decelerating, the *EGO* should decelerate accordingly to maintain a safe distance and avoid a collision.

In the subsequent chapters, we illustrate the results of the presented ALKS safety models and scenarios.

5.3 Results

In this section, we present the results of our experimental evaluation using Esmini as a simulation environment, the defined ALKS safety model as our systems under test, and the scenarios depicted in Section 5.2 as our use cases. We initially started our evaluation process by conducting 41 unit tests to ensure the correctness of our Continuous Validation Setup (CVS). This provided the foundation for performing an extensive experimental evaluation. To demonstrate the practical applicability of our methodology, based on 29 initial scenarios, we automatically generated 977 variations and integrated four different ALKS safety models, resulting in a total of 3,908 simulations executed. This process included the automatic post-processing of results and validation. All scenarios are based on the ASAM OpenSCENARIO and OpenDRIVE standards, ensuring realistic and reproducible test conditions. We evaluate the correct behavior of the system under test every 0.1 seconds. Considering all scenarios executed and investigated, Table 3 shows the number of violations that were identified for each safety model. It should be noted that each violation was counted, meaning that, for example, a continuous violation of one second consists of ten consecutive violations. With respect to *Overspeeding*, the number of violations is significantly lower compared to the other violations due to the fact that speed limits are only present in the *ALKS R157 Test Scenario*.

Table 3 Overview of all violations identified for each ALKS controller safety model.

Regulation Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
78	10705	14082	222306
Reference Driver Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
63	23374	10789	37211
RSS Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
59	12904	5346	3804
FSM Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
76	9889	16694	141892

From these scenarios, we highlight the results obtained for four representative use cases to illustrate the effectiveness of our approach. The scenarios involve various maneuvers and interactions between the *EGO* vehicle, equipped with an ALKS controller, and other road users (*NPC1*). The maneuvers and expected behaviors for each scenario are detailed in Section 5.2, providing insight into how the *EGO* vehicle should maintain safety and compliance with the regulations.

The following subsection will present a detailed description of the *ALKS R157 Test Scenario*, along with the observed faults and corresponding explanations. This comprehensive evaluation underscores the robustness of our proposed methodology for continuous monitoring, fault detection, and explanation using ASP in automated and autonomous driving systems. Within the Appendix, the results for the three other single maneuver scenarios are highlighted.

5.3.1 ALKS R157 Test Scenario (Figure 2a)

Within this section, we want to highlight the main results obtained for the scenario and included maneuvers depicted in Figure 2a. Figure 3, 4, 5, and 6 display the simulation output at the top as well as the output of our continuous validation setup at the bottom,

including the explanation of any misbehavior. As shown in the legend of the plots, the blue line represents the speed profile of the *EGO* vehicle, whereas the green line represents the vehicle speed of *NPC1*. Additionally, the dashed purple and magenta lines illustrate the relative distance to *NPC1* and the minimum following distance, respectively. Furthermore, it is important to note that the minimum following distance is dependent on the *EGO* vehicle speed. The current speed limit is depicted in red. At the bottom, we present the output of our CVS, where *FALSE* indicates that a fault was detected and *TRUE* shows that the ALKS controller complies with the regulations. Moreover, in case of a fault, the area below the curve is filled with different patterns to explain the cause of the fault. It is noteworthy to highlight the differences between *No Stopping*, *No Collision Avoidance*, and *Insufficient Speed Adaption*.

Insufficient Speed Adaption: This fault is identified if the *EGO* and *NPC1* are traveling with similar vehicle speeds, no abrupt decelerations occur, and the relative distance to the *NPC1* is less than the minimum following distance.

No Stopping: This occurs when the *EGO* is approaching a road user in the same lane who is traveling at low speeds or is performing a stopping maneuver, and the relative distance is greater than the minimum following distance. Hence, stopping in front of the road user would be possible without initiating a collision avoidance maneuver.

No Collision Avoidance: In contrast to *No Stopping*, this misbehavior is identified if the relative distance to a road user in the same lane is less than the minimum following distance, indicating that a collision would occur in case no avoidance maneuver is performed.

Additionally, since we receive no feedback from the integrated ALKS controllers on whether stopping or collision avoidance is initiated, we assume it to be false. Therefore, with more advanced controllers, some of the identified misbehaviors might be resolved when such additional information is available. In the following, we will elaborate in more detail on what faults occurred for the different safety models of the ALKS controller.

Figure 3 shows the results for the *ALKS R157 Test Scenario* and the ALKS controller with the integrated *Regulation* safety model. From the beginning of the simulation until around 2.5 seconds, we identify no faults since *NPC1* is still driving on the adjacent lane. However, after *NPC1* performs an overtaking maneuver followed by a cut-in, the *EGO* vehicle needs some time to re-establish a sufficient distance to the vehicle ahead. This delay results in a fault indicating *Insufficient Speed Adaptation* until the appropriate distance is regained. Furthermore, we see that between seconds 7 and 9, the controller fails to respect the present speed limit. This issue recurs during the time frame from 11 to 17 seconds. Our approach correctly detected both of these occurrences. Additionally, within seconds 11 and 20, we observe another instance of *Insufficient Speed Adaptation*, caused by the decelerating *NPC1*. At the end of the simulation, when *NPC1* comes to a stop, the *No Collision Avoidance* flag is raised because the relative distance to the road user is less than the minimum following distance, and no avoidance maneuver is initiated.

Within Figure 4, we showcase the results for the *Reference Driver* safety model. Similar to the previous results, the *EGO* vehicle's controller initially struggled to establish a safe following distance after *NPC1* performed the cut-in maneuver, which is also detected by our CVS. Additionally, unlike the *Regulation* model, the *Reference Driver* model successfully maintains the minimum following distance during the deceleration phase of *NPC1*. This indicates an improvement in adaptive behavior and responsiveness to changes in the relative speed and distance of *NPC1*. Despite these improvements, the *Reference Driver* model still

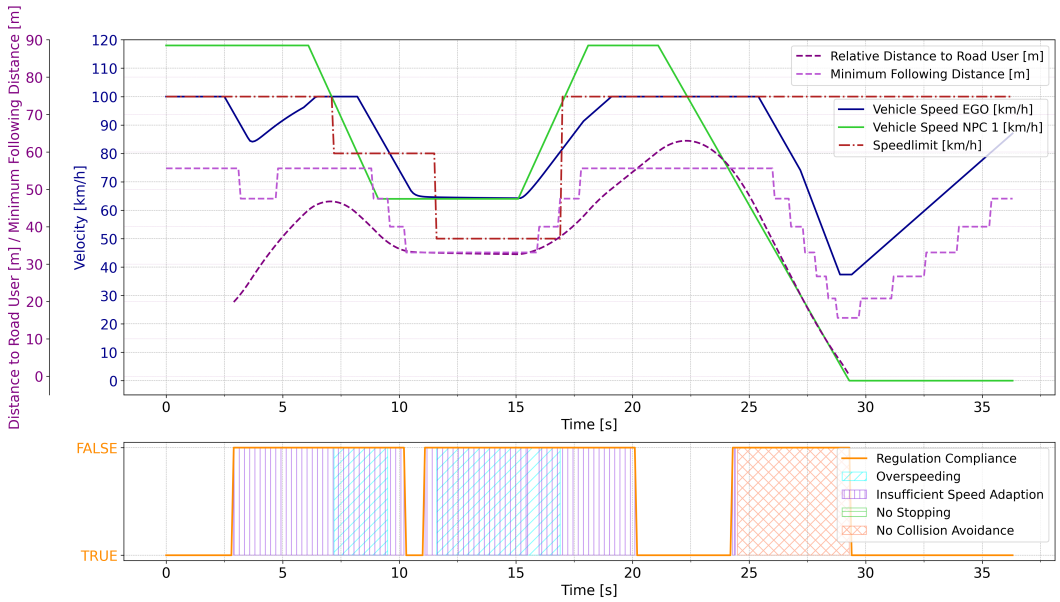


Figure 3 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *Regulation* safety model.

fails to comply with the speed limits present in the scenario. The *EGO* vehicle exceeds the speed limit at several points during the simulation, which is marked as overspeeding in the validation output.

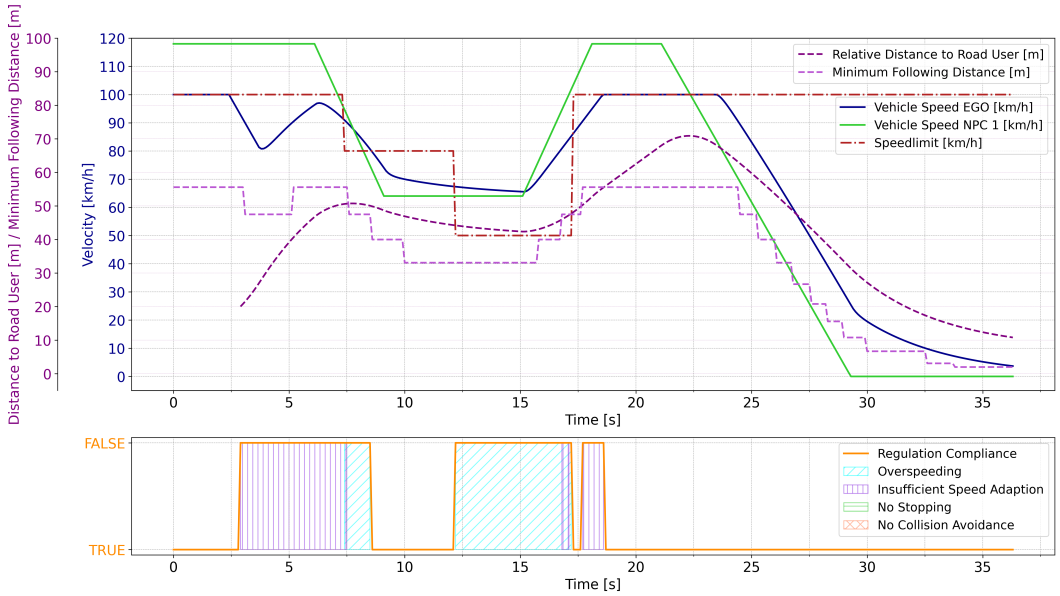


Figure 4 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *Reference Driver* safety model.

The outcome of our evaluation with the *RSS* safety model is depicted in Figure 5. As with the previous safety models, the ALKS controller fails to comply with the speed limits during certain time frames, i.e., between seconds 7 and 8 and from 12 to 17. Furthermore,

our CVS indicates a *Insufficient Speed Adaption* fault at the beginning of the simulation. Close to the end of the scenario, a *No Stopping* violation was identified. This stems from the fact that *NPC1* performs a stopping maneuver, and the *EGO* vehicle does not adapt to this reasonably. The *RSS* safety model demonstrates improvements over the previous safety models with respect to responsiveness in attempting to maintain a safe following distance, it still exhibits significant issues with speed limit compliance. However, as stated previously, those safety models are experimental implementations, and different outcomes are expected for more mature releases.

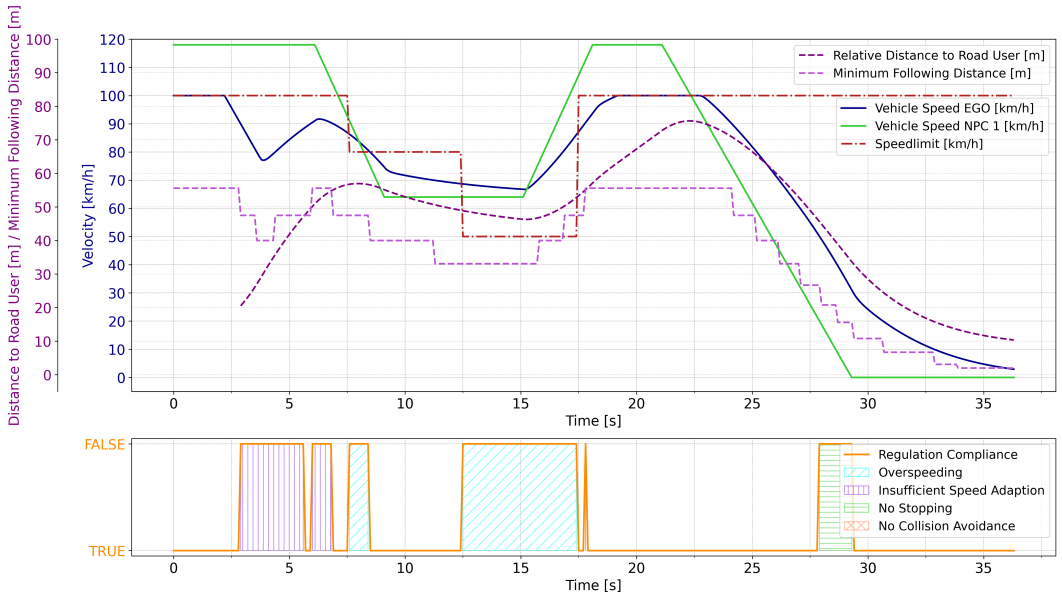


Figure 5 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *RSS* safety model.

The results depicted in Figure 6 illustrate the outcomes for the *FSM* safety model. There are noticeable similarities to the behavior of the *Regulation* safety model. We can observe that, in this simulation, the controller fails to adhere to the speed limits during specific periods. Additionally, it struggles to maintain a safe following distance to *NPC1* at several points, indicating that the vehicle does not consistently comply with the regulations. Furthermore, at the end of the simulation, the *No Collision Avoidance* fault is raised again, demonstrating that the *FSM* safety model reacts too late to the stopping maneuver performed by the *NPC1*.

The evaluation of our proposed methodology for continuous monitoring, fault detection, and explanation leveraging ASP was conducted through a series of unit tests and extensive experimental simulations using Esmini as a simulation environment. Initially, 41 unit tests were performed to ensure the correctness of our Continuous Validation Setup (CVS), all of which were successfully passed, providing a robust foundation for further testing.

Subsequently, 3,908 scenarios were executed, derived from 29 initial scenarios, incorporating four distinct ALKS safety models: *Regulation*, *Reference Driver*, *RSS*, and *FSM*. Our methodology was applied to each model to assess their compliance with UN R157 regulations and their overall performance under simulated driving conditions. Overall, our results demonstrate the successful identification of misbehaviors during various simulation periods, as depicted by the simulation data illustrated in the respective figures.

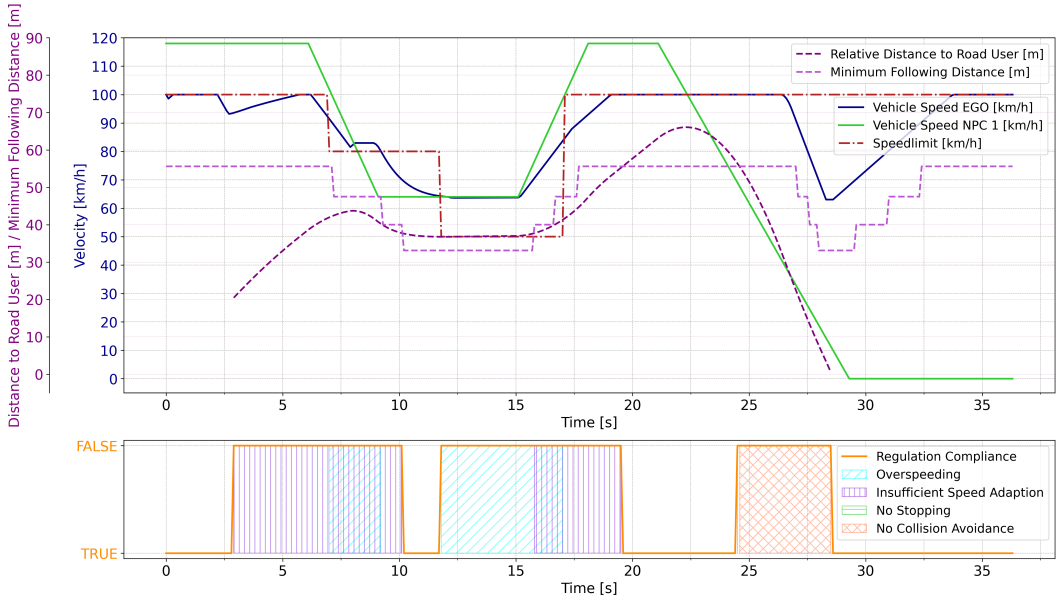


Figure 6 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *FSM* safety model.

6 Conclusion

In this paper, we proposed a novel methodology for continuous monitoring, fault detection, and explanation of automated and autonomous driving systems using Answer Set Programming (ASP). Our approach integrates ASP with data from simulation environments to identify discrepancies between expected and actual vehicle behavior, providing explanations for detected faults. Through a series of 41 unit tests, we validated the correctness of our Continuous Validation Setup (CVS). Furthermore, we conducted an extensive experimental evaluation within a simulation framework using Esmini as an environment simulator, encompassing 3,908 scenarios derived from 29 initial scenarios. These scenarios, in combination with four distinct ALKS safety models, were used to test our methodology's effectiveness in identifying faults during runtime. Our results demonstrated the effectiveness of our approach in identifying misbehaviors and compliance issues with UN Regulation No. 157 under various simulated driving conditions. The successful detection and explanation of faults, which can further serve as input to fault-adaptive control strategies and mitigation measures, highlights the potential of our methodology to enhance the reliability and safety of automated and autonomous driving systems. Future work will focus on extending our approach to more complex scenarios and integrating additional regulations and general driving knowledge. Additionally, challenges for real-world deployment, such as handling noisy signals and ensuring real-time performance within vehicles, will be addressed.

References

1 William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-642-55481-0.

2 Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-03754-2_2.

3 United Nations Economic Commission for Europe. Un regulation no 157 – uniform provisions concerning the approval of vehicles with regards to automated lane keeping systems [2021/389], March 2021. URL: http://data.europa.eu/eli/reg/2021/389/oj.

4 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi:10.1017/S1471068418000054.

5 Leilani Hendrina Gilpin. *Anomaly Detection Through Explanations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2020.

6 Ledio Jahaj, Lorenz Klampfl, and Franz Wotawa. Knowledge-based monitoring for checking law and regulation compliance. In Hamido Fujita, Richard Cimler, Andres Hernandez-Matamoros, and Moonis Ali, editors, *Advances and Trends in Artificial Intelligence. Theory and Applications*, pages 491–502, Singapore, 2024. Springer Nature Singapore. doi:10.1007/978-981-97-4677-4_40.

7 Nidhi Kalra and Susan M. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016. doi:10.1016/j.tra.2016.09.010.

8 Lorenz Klampfl, Florian Klück, and Franz Wotawa. Using genetic algorithms for automating automated lane-keeping system testing. *Journal of Software: Evolution and Process*, 36(3):e2520, 2022. doi:10.1002/smr.2520.

9 Lorenz Klampfl and Franz Wotawa. Identifying critical scenarios in autonomous driving during operation. In *Artificial Intelligence. ECAI 2023 International Workshops*, volume 1947 of *Communications in Computer and Information Science*, pages 156–172, Cham, 2024. Springer. Trustworthy AI for Safe & Secure Traffic Control in Connected & Autonomous Vehicles: TACTFUL 2023. doi:10.1007/978-3-031-50396-2_9.

10 Florian Klück, Martin Zimmermann, Franz Wotawa, and Mihai Nica. Genetic algorithm-based test parameter optimization for adas system testing. In *Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 418–425, 2019. doi:10.1109/QRS.2019.00058.

11 Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Safety*, 4:15–24, April 2016. doi:10.4271/2016-01-0128.

12 Konstantinos Mattas, Michail Makridis, George Botzoris, Akos Kriston, Fabrizio Minarini, Basil Papadopoulos, Fabrizio Re, Greger Rognelund, and Biagio Ciuffo. Fuzzy surrogate safety metrics for real-time assessment of rear-end collision risk. a study based on empirical observations. *Accident Analysis & Prevention*, 148:105794, 2020. doi:10.1016/j.aap.2020.105794.

13 Fabian Schuldt, Andreas Reschka, and Markus Maurer. *A Method for an Efficient, Systematic Test Case Generation for Advanced Driver Assistance Systems in Virtual Environments*, pages 147–175. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-61607-0_7.

14 Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars, 2018. arXiv:1708.06374.

15 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Out of sight but not out of mind: An answer set programming based online abduction framework for visual sensemaking in autonomous driving. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1879–1885. International Joint Conferences on Artificial Intelligence Organization, July 2019. doi:10.24963/ijcai.2019/260.

16 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Commonsense visual sensemaking for autonomous driving – on generalised neurosymbolic online abduction integrating vision and semantics. *Artificial Intelligence*, 299:103522, 2021. doi:10.1016/j.artint.2021.103522.

17 Franz Wotawa. *Testing Autonomous and Highly Configurable Systems: Challenges and Feasible Solutions*, pages 519–532. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-31895-0_22.

18 Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices*, pages 518–529, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-55789-8_45.

19 Franz Wotawa and David Kaufmann. Model-based reasoning using answer set programming. *Applied Intelligence*, 52(15):16993–17011, 2022. doi:10.1007/s10489-022-03272-2.

20 Franz Wotawa, Bernhard Peischl, Florian Klück, and Mihai Nica. Quality assurance methodologies for automated driving. *Elektrotechnik & Informationstechnik*, 135(4–5), 2018. doi:10.1007/s00502-018-0630-7.

7 Appendix

In this section Table 4-6 showcase the results obtained for the scenarios depicted in Figure 2b, 2c, and 2d.

Table 4 Results obtained for the *Cut-In (Right)* & *Deceleration* scenario.

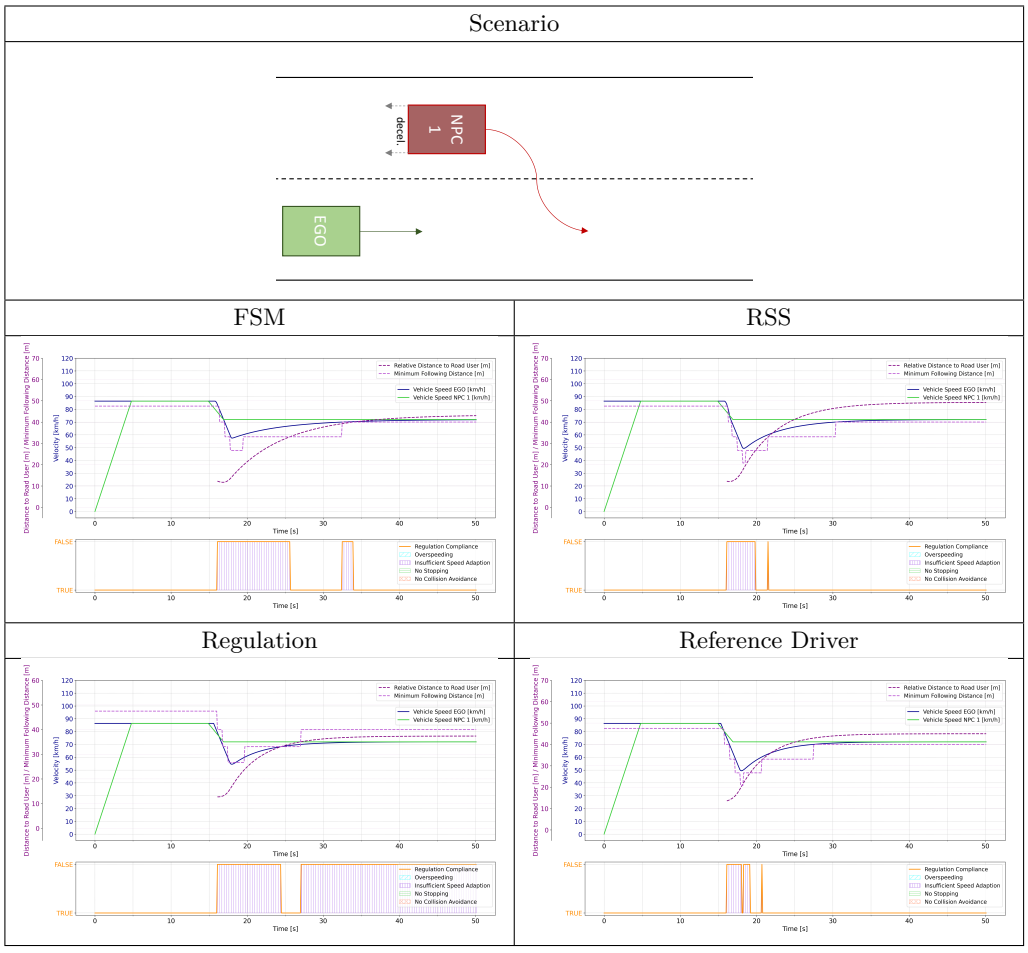


Table 5 Results obtained for the *Same Lane & Deceleration* scenario.

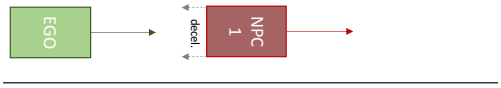
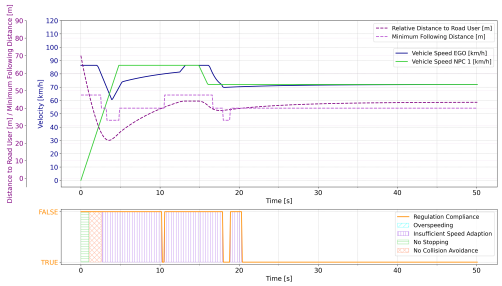
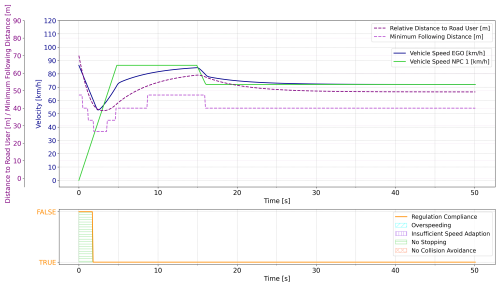
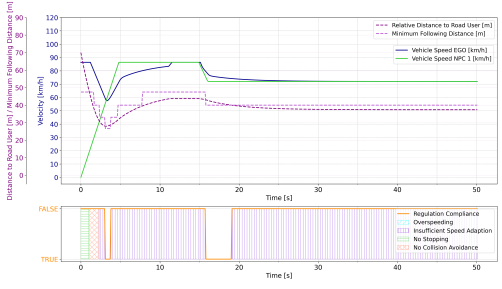
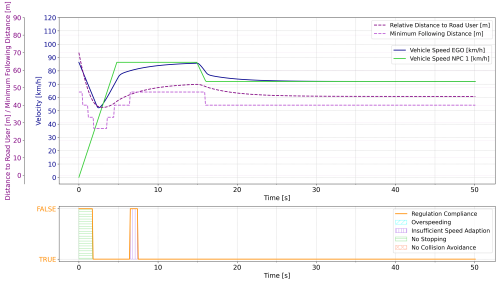
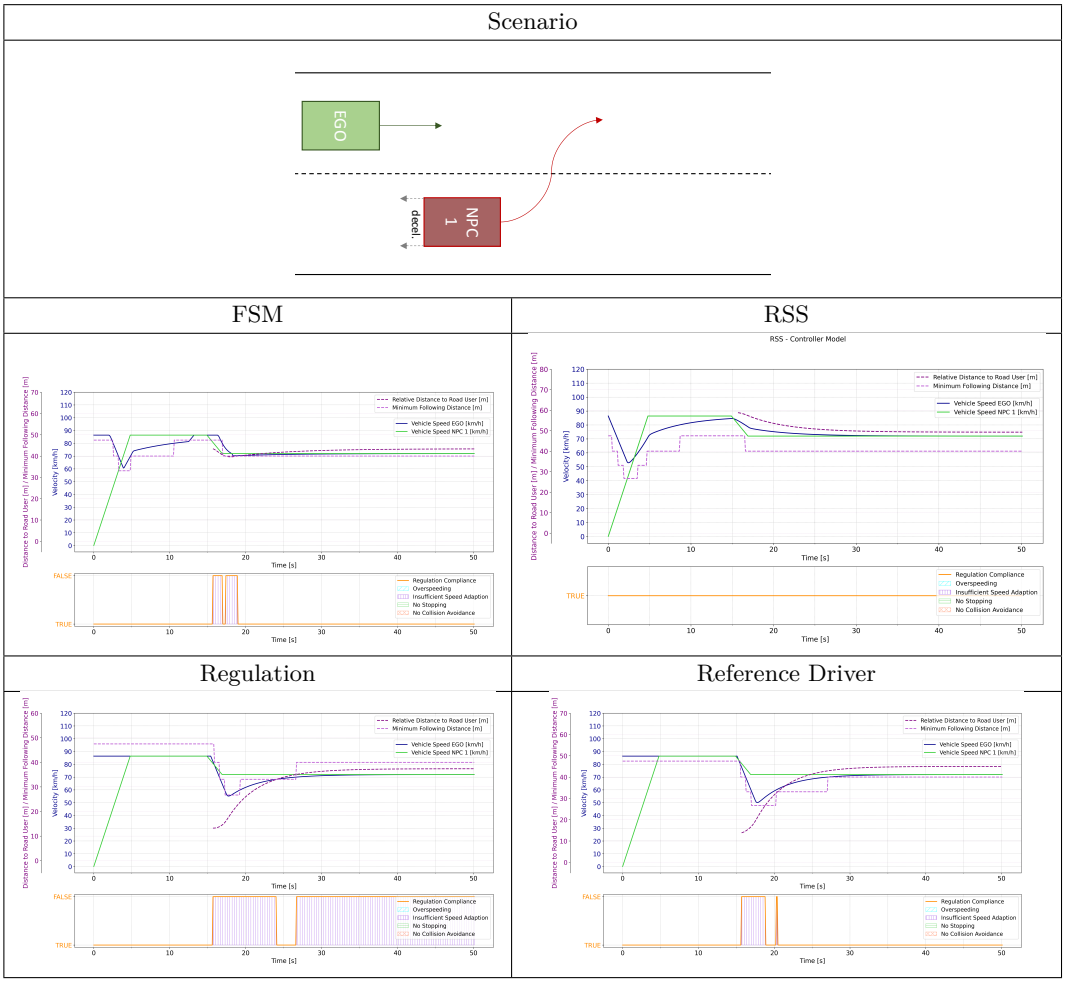
Scenario	
	
FSM	RSS
	
Regulation	Reference Driver
	

Table 6 Results obtained for the *Cut-In (Left)* & *Deceleration* scenario.



Leveraging Causal Information for Multivariate Timeseries Anomaly Detection

Lukas Heppel
University of Würzburg, Germany
German Aerospace Center (DLR), Lampoldshausen, Germany

Andreas Gerhardus
German Aerospace Center (DLR), Jena, Germany

Ferdinand Rewicki
German Aerospace Center (DLR), Jena, Germany

Jan Deeken
German Aerospace Center (DLR), Lampoldhausen, Germany

Günther Waxenegger-Wilfing
University of Würzburg, Germany
German Aerospace Center (DLR), Lampoldhausen, Germany

Abstract

Anomaly detection in multivariate timeseries is used in various domains, such as finance, IT, or aerospace, to identify irregular behavior in the used applications. Prior research in anomaly detection has focused on estimating the joint probability of all variables. Then, anomalies are scored based on the probability they receive. Thereby, the variables' dependencies are only considered implicitly. This work follows recent work in anomaly detection that integrates information about the causal relations between the variables in the timeseries into the detection mechanism. The causal mechanisms of the variables are then used to identify anomalies. An observation is identified as anomalous if at least one of the variables it contains deviates from its regular causal mechanism. These regular causal mechanisms are estimated via the conditional distribution of a variable given its causal parent variables, i.e., the variables having a causal influence on a variable. We further develop previous work by gathering information about the causal parents of the variables by applying causal discovery algorithms adapted to the timeseries setting. We apply Conditional Kernel Density Estimation and Conditional Variational Autoencoders to estimate the conditional probabilities. With this causal approach, we outperform methods that rely on the joint probability of the variables in our synthetically generated datasets and the C-MAPPS dataset, which provides simulation data of turbofan engines. Moreover, we investigate the causal approach's inferred scores on the C-MAPPS dataset to gather insights into the measurements responsible for the prediction of anomalies. Furthermore, we investigate the influence of deviations from the true causal graph on the anomaly detection performance using synthetic data.

2012 ACM Subject Classification Computing methodologies → Anomaly detection

Keywords and phrases Anomaly Detection, Causal Discovery, Multivariate Timeseries

Digital Object Identifier 10.4230/OASIcs.DX.2024.11

1 Introduction

Sensors generate data throughout the operational lifespan of assets across various domains, including aerospace and manufacturing [9]. These assets, such as turbofan engines, are susceptible to deterioration and spontaneous faults [26]. Consequently, one of the primary objectives when operating these assets is to assess their condition. Once an asset has reached a certain point of degradation, it is necessary to maintain or replace it to ensure the safety of ongoing operations [10, 9]. The continuous monitoring of assets with the assistance of sensors



© Lukas Heppel, Andreas Gerhardus, Ferdinand Rewicki, Jan Deeken, and Günther Waxenegger-Wilfing;
licensed under Creative Commons License CC-BY 4.0



generates multivariate timeseries data. In recent years, data-driven approaches have emerged to monitor the health status of these assets, which is coupled with detecting anomalies in multivariate timeseries [36, 9]. In anomaly detection (AD), the objective is to identify data samples that deviate from the prevailing patterns within the dataset [5]. An increase in the number of anomalous observations may indicate asset degradation [9].

Several approaches exist to tackle AD in multivariate timeseries, which consider the multivariate timeseries holistically and only implicitly consider the variables' dependencies. These approaches, for example, rely on Autoencoders (AEs) and variants thereof, such as the Variational Autoencoder (VAE), which estimate the joint probability of individual observations or of observations in a row to describe the regular behavior of the system under consideration [9, 37]. Inference with these methods is then done by assessing the likelihood of given observations. Lower probabilities indicate that an observation could be anomalous [37]. However, when modeling the joint probability of the individual observations directly, we do not explicitly consider the relations of the variables. Recent work on AD in multivariate timeseries explicitly considers the relations among the individual sensors uni-variate timeseries [4, 35]. However, the considered relations are not the causal relations of the underlying system. Thus, recent work by Yang et al. [33] integrates the causal relations amongst the component timeseries into AD. Using the causal relations, the estimation of the joint probability of a sample is factorized into a product of the conditional distributions of the variables given their causal parents, i.e., the variables causing a variable. The conditional distributions are used to characterize the causal mechanisms of the variables. In the causal approach, an anomaly is present if the causal mechanism is disturbed at least on one variable of the multivariate timeseries [33].

However, the causal relations among the variables are often unknown. One approach to obtaining the causes and effects present in a process is to intervene with a system to conduct experiments actively. This is often impracticable since it can be time-consuming, cost-intensive, or requires domain knowledge. However, we can gather multivariate timeseries data from the system through observation [23]. Then, we can reconstruct the timeseries graph embodying the causal parents of the variables based on observational data via Causal Discovery (CD) [24]. The timeseries graph contains the variables at the different timesteps as nodes, and directed edges indicate the causal relations. Thereby, an edge is directed from the causal parent to the child.

In this work, we further develop the causal approach by Yang et al. [33] by applying constraint-based CD algorithms tailored for the multivariate timeseries case, PCMCI [25], and PCMCIplus [22]. Using the reconstructed timeseries graph, we estimate the conditional distribution given its causal parents for every variable. For inference, we use the ensemble of estimates to check whether the causal mechanism of at least one variable is deteriorated. To analyze the effect of the causal relations used, we generate synthetic data that we model as a structural causal process. This way, we know the ground truth causal relations and can evaluate the effects of using relations that differ from the correct ones. Furthermore, we benchmark the causal approach on the publicly available C-MAPPS dataset, which provides simulations of turbofan engines. In addition, we use the inferred anomaly scores from the causal approach to gather insights into the anomalies' emergence.

Our contributions are as follows:

- We develop the causal approach by Yang et al.[33] further by applying CD algorithms tailored for the multivariate timeseries case.
- We use synthetic data to examine the impact of deviations from the true causal graph on the anomaly detection capability.
- We benchmark the causal approach on the publicly available C-MAPPS dataset.

The remainder of this paper is structured as follows: In Section 2, we give an overview of approaches in anomaly detection for timeseries data and introduce the necessary concepts our approach builds upon in Section 3. Afterward, in Section 4, we describe the approach in detail, and in Section 5, we explain our experiments, which we use to evaluate our models. In Section 6, we discuss this work, point out directions for future work, and end this paper in Section 6 with our conclusion.

This paper is the result of work carried out as part of Lukas Heppel's Master's thesis, which was supervised by Günther Waxenegger-Wilfing.

2 Related Work

This section reviews related work on anomaly detection, focusing on detecting anomalies in multivariate timeseries.

Methods for AD comprise distance-based techniques such as the k-nearest Neighbour (kNN) algorithm. In addition, statistical approaches exist to tackle detecting anomalies in timeseries data [30]. Furthermore, Support Vector Machines (SVMs) are trained to classify anomalies in timeseries data [3, 18, 1]. Moreover, anomalies are detected using Isolation Forests (IFs) [11]. Additionally, Yaacob et al. [32] fit an Auto-Regressive Integrated Moving Average (ARIMA) on the timeseries. Afterward, observations are classified based on the error between the prediction of the ARIMA model for the current value and the actual observed value.

In recent years, deep learning methods have been applied to identify anomalies in multivariate timeseries data [5]. Several approaches use Autoencoders (AEs) to detect anomalies [19, 9, 20]. Jakubowski et al. [9] and Reddy et al. [19] train AEs to reconstruct the input data. Anomaly scores are then obtained as the error between the input and its reconstruction. Other AE variants, besides the vanilla AE, are also used to detect anomalies. For example, Variational Autoencoders (VAEs) [9, 31] allow obtaining the reconstruction probability of the data points and then computing anomaly scores. Zong et al. [37] introduce Deep Autoencoding Gaussian Mixture Model (DAGMM) for AD in multivariate timeseries. This approach consists of an AE, which provides lower-dimensional representations of the input, i.e., the latent representation. The latent representation and the corresponding reconstruction error are then used to estimate the density of a sample using the estimation network, which is trained as a Gaussian Mixture Model (GMM). In addition to the aforementioned AE variants, where the encoder and decoder consist of Feedforward Neural Networks, the approach of Park et al. [14] combines LSTMs with VAEs to encode the temporal dependencies of the timeseries. However, these methods implicitly consider the dependencies among the variables and ignore explicit knowledge, which is sometimes available. In addition, these approaches cannot trivially be used to explain an anomaly since they only provide one score/classification per observation.

Further approaches arise that strive to include knowledge of the dependencies of the variables contained in the multivariate timeseries [4, 6]. Dai et al. [4] use normalizing flows to estimate the joint probability of all variables in the multivariate timeseries. They incorporate a Directed Acyclic Graph (DAG) into the probability estimation to factorize the joint probability into a product of conditional densities. The DAG used for factorizing the joint probability is optimized with the normalizing flow. In addition, Deng et al. [6] apply graph attention-based forecasting of future sensor values for AD while relying on graph structures that represent as nodes the sensors, which are connected by edges with the k nodes they have the most similar timeseries with. However, the variables' relations used in these approaches are not necessarily causal relations.

Therefore, Yang et al. [33] explicitly consider the relations among the variables. Thus, they reconstruct the causal relations present in the multivariate time via the CD algorithm PC and score-based CD algorithms. Based on the causal relations, they estimate the conditional distribution of each variable, given its causal parents, via CVAEs. For the inference, they predict an anomaly if one variable's model predicts an anomaly smaller than a given threshold.

Our approach differs from the methodology proposed by Yang et al. [33] in detecting the causal relations and the used density estimation techniques. To this end, we rely on the constraint-based approaches PCMCI [25] and PCMCIplus [22], which are adaptions for the timeseries setting and show superior performance compared to the PC algorithm [28], as shown by Runge et al. [25, 22]. Moreover, we examine in greater detail the impact of deviations from the true graph structure on the AD performance of the causal approach.

3 Preliminaries

This section introduces the necessary concepts this paper builds upon.

3.1 Reconstruction-based AD

This work is concerned with detecting anomalies in the multivariate timeseries setting. Thus, we define a multivariate timeseries as $x = (x^1, \dots x^d) \in \mathbb{R}^{T \times d}$, which is an ordered sequence of d univariate timeseries $x^i \in \mathbb{R}^T$, where each describes a feature x^i over a period of length T. We denote the value of feature i at time t by $x_t^i \in \mathbb{R}$. Furthermore, $x_t = (x_t^1, \dots, x_t^d) \in \mathbb{R}^d$ is the vector describing the value of all features at time t.

Autoencoders (AEs) are used to identify anomalies [9]. The AE architecture consists of an encoder and a decoder. The encoder maps an input vector $x \in \mathbb{R}^{d_{input}}$ into a latent representation $z \in \mathbb{R}^{d_{latent}}$. In most cases, AEs apply a bottleneck, i.e., the dimension $d_{latent} < d_{input}$ of the latent space is smaller than the dimension of the input. Introducing a bottleneck pushes the AE to learn meaningful latent representations from the input. Next, the decoder transforms the latent representation z into a reconstruction $\hat{x} \in \mathbb{R}^{d_{input}}$ of the input dimension. We train the AE to minimize the error between the input x and the reconstruction $\hat{x}$ using the Mean Squared Error (MSE). For inference, the reconstruction error of a sample can be used as an anomaly score, assuming that anomalous data differs from the known training data and, therefore, cannot be properly reconstructed.

The Variational Autoencoder (VAE) is a variant of the AE that estimates the distribution of a given dataset [13]. Thereby, the assumption is made that the generative process of the data points includes continuous latent variables z. To this end, the VAE's encoder approximates the distribution $q(z \mid x)$, and the decoder estimates the distribution $p(x \mid z)$. We can then formulate the variational lower bound of $log\ p(x_i)$, which is also referred to as evidence lower bound (ELBO), as follows [13]:

$$log\ p(x_i) \geq \mathbb{E}_{z \sim q(z|x_i)}[log\ p(x_i \mid z)] - D_{KL}(q(z \mid x_i) \mid\mid p(z)) \tag{1}$$

Thereby, D_{KL} represents the Kullback-Leibler (KL) divergence [34]. The initial term represents the KL divergence between the encoder distribution $q(z \mid x_i)$ and the prior $p(z)$, which we assume to be a standard normal distribution $\mathcal{N}(0, 1)$. Consequently, the learned posterior distribution is regularized to be close to a standard normal distribution. The second term is the expected negative log-likelihood of the input given the latent representation. We compute the expectation by sampling a number of latent representations z_i for which the decoder network provides the reconstructions. These reconstructions are then used to derive the parameters of the decoder distribution, which we assume to be Gaussian. The VAE can then be trained by minimizing the negative ELBO.

To detect anomalies, we are then required to infer the likelihood of a sample that should be assessed whether it's anomalous. We assume that anomalies obtain smaller likelihoods since they deviate from the distribution of the normal samples. Throughout this work, we measure a sample x_i's density $p(x_i)$ by the decoder's distribution based on the reconstructions resulting from the latent samples $z_i \sim p(z \mid x_i)$.

3.2 CVAE

The Conditional Variational Autoencoder (CVAE) [27] is an extension of the VAE that allows for conditional inputs. The CVAE allows us to guide its generative process via attributes, e.g., in image generation [2]. Thus, the CVAE estimates the conditional log-likelihood of the data. Consequently, the encoder is extended to approximate $q(z \mid x, c)$. Moreover, the decoder is extended to model $p(x \mid z, c)$. Then, to estimate the conditional log-likelihood, the variational lower bound of a data point x_i given c_i is defined as follows [27]:

$$log\ p(x_i \mid c_i) \geq \mathbb{E}_{z \sim q(z \mid x_i, c_i)}[log\ p(x_i \mid z, c_i))] - D_{KL}(q(z \mid x_i, c_i) \mid\mid p(z \mid c)) \tag{2}$$

We assume the prior $p(z \mid c) = \mathcal{N}(0, 1)$ to be a standard Normal distribution. The CVAE can then be trained by minimizing the negative variational lower bound.

3.3 Causal Discovery

The AD approach proposed in this work requires information about the causal relations underlying the observed system, which is the origin of the multivariate timeseries data. However, the causal relations are often unknown and need to be discovered. We could intervene and actively conduct experiments to gather information about the causal relationships in the system under consideration. However, active experimentation is often impossible because it is time-consuming, cost-intensive, demands expert knowledge, or is infeasible, e.g., in our case, where we do not have the knowledge to work with the simulation software providing our dataset. Nevertheless, the systems under consideration can often be observed to obtain multivariate timeseries data [23].

In CD in the timeseries setting, we aim to reconstruct the underlying timeseries graph that embodies the temporal dependency structure of a given dynamic system. For this purpose, we assume a *discrete-time structural causal process* $x_t = (x_t^1, \dots x_t^d)$, which is defined as follows [22]:

$$x_t^i = f_i(pa(x_t^i), \eta_t^i), \tag{3}$$

with f_i denoting arbitrary measurable functions that non-trivially depend on their arguments. Moreover, $pa(x_t^i)$ are the causal parents of the variable x_t^i, and η_t^i represents noise, which is assumed to be independent between the variables and the timesteps.

The *timeseries graph* $G = (V, E)$ is defined as follows:

- The set of nodes V represents the variables x_t^i at the different timesteps t.
- E is the set of edges containing an edge $x_{t-\tau}^j \to x_t^i$ iff $x_{t-\tau}^j \in pa(x_t^i)$

Furthermore, we define the variables a variable x_t^i causally depends on, as its causal parents $pa(x_t^i) \subset (x_t, x_{t-1}, \dots)$. Thereby, we refer to links with $\tau = 0$ as contemporaneous links and those with $\tau > 0$ as lagged links. In the case $i = j$, we refer to such links as autodependency. In addition, we assume the graph to be acyclic. Moreover, we assume the case of Causal Stationarity, which means that the causal relationships are assumed to be invariant in time, i.e., if there exists a link $x_{t-\tau}^j \to x_t^i$ for a t, then we assume $x_{t'-\tau}^j \to x_{t'}^i$ for all $t' \neq t$.

Alternatively, different parts of a time series can be represented by different graphs to model non-stationarity [24]. Theoretically, the graph is infinite. However, we consider the graph only up to a maximum time lag τ_{max}.

The predominant methodologies employed in the field of CD can be broadly classified into three main categories: score-based, asymmetry-based, and constraint-based [24]. We are concerned with applying constraint-based methods to detect causal relations in multivariate timeseries. Score-based and asymmetry-based approaches are still in their infancy for timeseries [24]. One popular approach for CD is the PC algorithm [28], originally developed for independent and identically distributed data.

The PC algorithm is the base for the PCMCI [25], and the PCMCIplus [22] algorithms, which we use for multivariate timeseries data in this paper. Constraint-based methods for CD, such as the PC algorithm, are conducted in two steps. In the initial phase, the graph's skeleton is reconstructed. This skeleton is an undirected graph that merely indicates whether two nodes are connected. In this phase, we employ an iterative approach to ascertain which pairs of variables are conditionally independent based on tests of the form:

$$H_0 : x^i \perp\!\!\!\perp x^j \mid x^k \qquad\qquad H_1 : x^i \not\perp\!\!\!\perp x^j \mid x^k,$$

thereby H_0 states the conditional independence of x^i and x^j given x^k, whereas H_1 states their conditional dependence. In the second phase, the skeleton's links are oriented, i.e., the directions for the undirected edges are determined. Therefore, time provides an orientation for lagged links, as causes precede effects. The orientation of contemporaneous links ($\tau = 0$) is not a straightforward process and is determined by applying additional rules. In general, constraint-based approaches can only detect the contemporaneous graph's Markov equivalence class [23]. The Markov equivalence class is the set containing all DAGs that embody the same conditional independence [16]. Thereby, the directions of the edges must not be the same in all DAGs of the Markov equivalence class. Thus, links between variables x_t^i and x_t^j where both directions, i.e., $x_t^i \to x_t^j$ and $x_t^i \leftarrow x_t^j$ are present in at least one DAG of the equivalence class, are represented as unoriented links in the resulting graph.

Since we can only measure statistical dependencies from data in the constraint-based approaches, we need several assumptions to interpret the graph structure obtained from constraint-based approaches as the timeseries graph, which describes the causal relations. We make the following assumptions to be able to apply the later discussed CD algorithms:

- *Causal Sufficiency*: This assumption states that no other unobserved variables directly or indirectly influence any other set of observed variables [25, 29].
- The *Causal Markov condition* states that the connectivity from the causal structure imprints the marginal and conditional (in)dependencies into the observed distribution [24, 29].
- The *Faithfulness* assumption states that all conditional independencies result from the causal structure [24, 29].

Using the Causal Markov condition and the Faithfulness condition, the following equivalence holds [24]: $x^i \perp\!\!\!\perp x^j \mid x^k \iff x^i$ d-sep $x^j \mid x^k$. This equivalence relates the graph structure, i.e., the contained d-separations [24], to the conditional independencies detected in the data.

The PCMCI algorithm addresses issues of the conditional independence framework when applied to the timeseries setting, e.g., the PC algorithm is prone to high false positive rates when facing auto-correlation [25]. In addition to the aforementioned assumptions, the PCMCI algorithm relies on the assumption that no contemporaneous links exist ($\tau = 0$) since PCMCI cannot orientate these. However, when we cannot assume the absence of contemporaneous

links, we apply the PCMCIplus algorithm to detect contemporaneous graph structures up to the Markov equivalence class [22]. It is possible that some links remain unoriented, which indicates that both directions exist for this link in the Markov equivalence class. In addition, conflicts may arise when applying the orientation rules, in which case the links are marked accordingly. One advantage of constraint-based approaches for CD is that they can be combined with several variants of conditional independence tests based on the dependencies assumed in the observed data. Thereby, they test $x^i \perp\!\!\!\perp x^j \mid x^k$. In the following, we provide information on the conditional independence tests considered in this paper:

- The Partial correlation test [25] (ParCorr) assumes a linear additive noise model with Gaussian noise. The test consists of two steps. First, ordinary least square regressions for x^i and x^j on x^k are fitted. Afterward, the Pearson correlation test is applied to the residuals.
- The Robust Partial Correlation [25] (RobParCorr) assumes the existence of a linear additive noise model with Gaussian noise and that the observed variables emerge thereof by component-wise transformation. RobParCorr can be applied when linear dependencies exist among the variables. In addition, RobParCorr is suited for different marginal distributions. Thus, RobParCorr extends ParCorr by first transforming the marginal distributions of the variables to standard normal marginals. Afterward, ParCorr is applied.

When applying the CD algorithms, we must determine the appropriate conditional independence test. To ascertain the dependency types among the variables, we visually inspect the scatter plots of pairs of variables with varying time lags. Furthermore, we visually inspect the marginal densities of the variables and the joint densities of the pairs of variables. Based on these observations, the most appropriate test for using the CD algorithm can be selected. Furthermore, we need to determine the minimum time lag τ_{min} and the maximum time lag τ_{max}. The choice of τ_{min} has to be made based on assumptions on the data. Thereby, the case that it is necessary to set $\tau_{min} = 0$ can occur when the sampling rate of the data is smaller than the rate at which the effects occur in reality. Domain knowledge is thereby a good help. The selection of τ_{max} is contingent upon the maximum anticipated time lag within the data. Consequently, the optimal value for τ_{max} can be determined by applying domain knowledge or by investigating the lagged dependencies of the pairs of variables, which refers to the lagged cross-correlation function between the pairs of variables, i.e., by testing the unconditional independence of the pairs of variables for varying time lags [25].

4 Methodology

This section describes the methodology we evaluate in this paper to detect anomalies in multivariate timeseries. First, we describe the decomposition of the task by incorporating the causal relations among the variables as described by Yang et al. in [33]. We explain how to obtain causal relations by conducting CD. Then, we outline the model fitting process for estimating the necessary (conditional) distributions. Finally, we present the inference of anomaly scores from the estimated models.

4.1 Factorization using Causal Relations

We assume $x = (x^1, \ldots, x^d) \in \mathbb{R}^{T \times d}$ to be a multivariate timeseries as defined in Section 3. Furthermore, we assume x to represent the regular behavior of the considered system, i.e., not containing anomalous points. Then, the task is to detect anomalous observations occurring

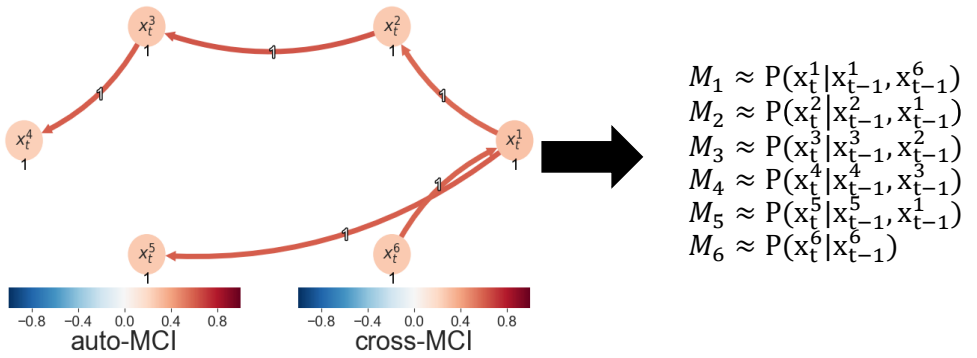


Figure 1 Illustration of how we obtain the models estimating the conditional distributions representing the causal mechanism of the variables from the process graph, which summarizes a timeseries graph resulting from a CD. The links indicate the causal parent relations and the time lag for the corresponding relation is noted at the link. For the autodependencies, the time lag is depicted next to a node. The color of the links and the nodes indicate the value of the conditional independence tests.

after time T. To detect anomalies x_t with $t > T$, we aim to identify data points x_t, where at least one $x_t^i \in x_t$ deviates from the causal mechanism so that the observed x_t^i and its causal parents $pa(x_t^i)$ deviate from the conditional distribution $P(x_t^i \mid pa(x_t^i))$, which characterizes the ordinary causal relation between x_t^i and its causal parents $pa(x_t^i)$. The causal relations of the timeseries can be represented by the corresponding discrete-time structural causal process $x_t^i = f_i(pa(x_t^i), \eta_t^i)$, as described in Section 3. Using the causal relationships between the variables, the joint distribution can be computed according to the Markov factorization as follows [16, 33]:

$$P(x_t) = \prod_{x_t^i \in x_t} P(x_t^i \mid pa(x_t^i)) \tag{4}$$

Thus, we decompose the task of estimating the joint distribution $P(x_t)$ into estimating one model M_i for the conditional distribution $P(x_t^i \mid pa(x_t^i))$ per variable $x_t^i \in x_t$ given its causal parents $pa(x_t^i)$. Figure 1 illustrates how we obtain the models M_i from an exemplary process graph, which summarizes a timeseries graph.

In some cases, we know these relations and can directly use them. In general, we do not know these relations. Therefore, we are concerned with retrieving the causal relations amongst the variables $x_t^i \in x_t$. To this end, we apply CD to obtain the causal relations. We execute a data analysis of the data to be discovered. Based on the dependency types of the variables, their joint and marginal distributions, their τ_{min}, and τ_{max}, and their lagged cross-correlations, we can select the adequate CD algorithm in combination with the proper conditional independence test. The result of the CD is then the timeseries graph representing the causal dependencies among the variables $x_t^i \in x_t$. However, the result of the PCMCIplus algorithm potentially provides unoriented links since the algorithm can only detect the Markov Equivalence class of contemporaneous links. Furthermore, we can encounter conflicting links where the application of the orientation rules is impossible. Thus, we consider the following post-processing options for the CD results to obtain the causal parents for each variable:

- One approach only considers the directed links in the detected timeseries graph. The unoriented and the conflicting links are not further considered. We refer to this technique as partial factorization throughout this paper.

- A further approach would be to consider using both possible directions of an unoriented link $x_t^i - x_t^j$, i.e., $x_t^i \to x_t^j$ and $x_t^i \leftarrow x_t^j$. The conflicting links are also not considered further.

4.2 Inference of Anomaly Scores

After discovering the causal relationship of the variables, we train one model M_i for every variable $x_t^i \in x_t$ to estimate the corresponding conditional probability distribution $P(x_t^i \mid pa(x_t^i))$. We follow Yang et al. [33] and employ a CVAE to estimate the respective density functions. Additionally, we consider Conditional Kernel Density Estimation (CKDE) [17]. In theory, further approaches can be integrated to estimate the conditional distributions. We estimate the conditional distributions $P(x_t^i \mid pa(x_t^i))$ using normal data samples $\{(x_t^i, c_t^i) \mid 1 \leq i \leq T\}$, with $c_t^i = pa(x_t^i)$ and T being the length of our training data.

Depending on the obtained timeseries graph, variables $x_t^i \in x_t$ may exist without causal parents $pa(x_t^i) = \varnothing$. This paper evaluates several methods to estimate these variables' distribution $P(x_t^i)$, i.e., the respective density functions. The following approaches are considered:

- We estimate the density $p(x_t^i)$ using a VAE. However, we encounter that variables without causal parents lead to one-dimensional inputs, so we cannot apply dimensionality reduction in the VAE.
- A further approach is to estimate $p(x_t^i)$ for a variable x_t^i without causal parents using Kernel Density Estimation.

To estimate $p(x_t^i)$, the corresponding training data is defined as follows $\{x_t^i, \mid 1 \leq i \leq T\}$. In addition, if multiple variables x_t^i exist without causal parents, we can apply a VAE to estimate the joint probability of these variables. We train the respective CVAEs and the VAEs by minimizing the negative variational lower bounds, which are described in Section 3.

After training the models M_i to estimate (conditional) distributions on the training data, we use the ensemble of the models to infer anomaly scores for previously unseen data samples x_t, with $t > T$.

In the following, we describe the inference of anomaly scores from the ensemble of models M_i. For every variable x_t^i, we infer $log\ p(x_t^i \mid pa(x_t^i))$, i.e., the logarithm of the likelihood of the variable's current value x_t^i given the current observed values of its parents. For variables without causal parents, we infer $log\ p(x_t^i)$, the log-likelihood of the current value. We assume that the lower the detected likelihood, the more anomalous a variable's current value is considered to be. Thus, to obtain an anomaly score, where higher values indicate a higher likelihood of an anomaly, we use the negative log-likelihoods as the anomaly scores of the variables. Thus, the anomaly score s_t^i of model M_i for an observation x_t^i is defined as follows:

$$s_t^i = M_i(x_t^i) = \begin{cases} -log\ p(x_t^i) & \text{if } \ pa(x_t^i) = \varnothing, \\ -log\ p(x_t^i \mid pa(x_t^i)) \end{cases}$$

To obtain the final anomaly score s_t of the ensemble, we consider using the maximum of the scores from all models. Since we assume that one variable violating its causal mechanism is sufficient to mark the complete observation as an anomaly, we consider taking the maximum of all local anomaly scores as the final score:

$$s_t = \max\left\{s_t^i \mid x_t^i \in x_t\right\} \tag{5}$$

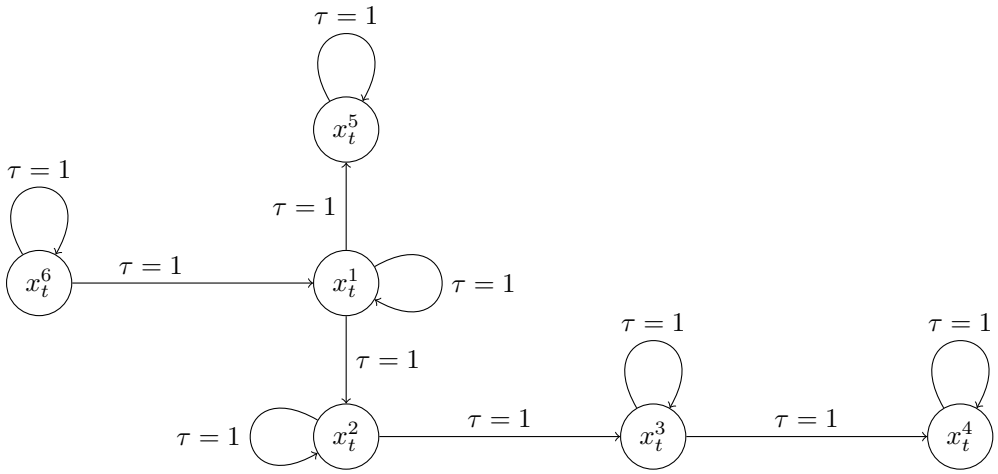


Figure 2 The ground truth process graph defining causal relations in the structural causal process to generate synthetic multivariate timeseries data. The coefficients for the causal dependencies are not depicted. E.g., $pa(x_t^2) = \{x_{t-1}^1, x_{t-1}^2\}$.

5 Experiments

This section presents the experiments conducted to evaluate the proposed methodology. To this end, the methodology is evaluated on synthetically generated data with a known ground truth graph structure. Furthermore, the impact of the applied graph structure on AD performance is examined. Finally, the approach is benchmarked on the publicly available C-MAPPS dataset [26].

5.1 Synthetic Data

We evaluate the causal approach on synthetically generated data. Thus, we define a structural causal process as a generalized additive model. The model is defined as follows [22]:

$$x_t^i = \eta_t^i + a^i x_{t-1}^i + \sum_{x_{t-\tau}^j \in pa(x_t^i) \setminus x_{t-1}^i} c_\tau^j f_\tau^j(x_{t-\tau}^j) \tag{6}$$

with variables $i \in \{1, \ldots, d\}$ and $\eta_t^i \sim \mathcal{N}(\mu_i, \sigma_i)$ denoting the additive Gaussian noise term of variable i at timestep t, which is sampled from a Gaussian distribution with mean μ_i and standard deviation σ_i. In addition, a^i represents the coefficient for the autodependency of variable i, and c_τ^j refers to the dependency coefficient of variable j over time lag τ. Moreover, $pa(x_t^i)$ denotes the set of causal parents $x_{t-\tau}^j$ of the variable i with the corresponding time lag τ and $f_\tau^j(x_{t-\tau}^j)$ corresponds to the dependency function.

Next, we define the ground truth causal relations of the model, i.e., for every variable x_t^i, its causal parents $pa(x_t^i)$. The process graph depicting the causal relations used for our datasets is depicted in Figure 2.

Furthermore, we use for all (auto)dependency coefficients 0.3, and we sample all variables' noise terms from $\mathcal{N}(0, 0.1)$. The applied dependency function is defined as $f(x) = (1 - 4e^{\frac{-x^3}{2}})x$. We use these parameters to generate the multivariate timeseries data to train the models based on regular data.

Moreover, we add anomalous datapoints into the test data. Therefore, we follow Yang et al. [33] and differentiate between the following types of anomalies:

- Intervention: We insert an anomaly at a variable i at time t. The anomaly is then propagated along the causal mechanism to the causal children variables j. In our case, we chose variable x^2.
- Measurement: We insert an anomaly at a variable i at time t. The anomaly is not propagated along the causal mechanism. We imitate a measurement error, i.e., the system works correctly, but, e.g., a sensor reported a wrong value. In our case, we chose variable x^6.
- Effect: We insert an anomaly at a variable i at time t. We chose a variable i, which does not have causal children. In addition, we remove the autodependency of this variable. In our case, we chose variable x^5.

Furthermore, we control the number of anomalies we insert into the test data. Thus, we randomly corrupt ten % of data points as anomalies. Thus, we transform normal data points into anomalous data points by defining a Uniform distribution $\mathcal{U}(2, 10)$, which we use to sample factors to scale a normal sample to be anomalous. We limit the resulting values of anomalous data to the range of the train data. We generate 7000 samples for model training, where we use 15 % for validation. In addition, we generate 3000 samples for testing.

For the causal approach, we apply Conditional Kernel Density Estimation (CKDE) or CVAEs to fit the conditional density of a variable x_i given the causal parents $pa(x_i)$.

Throughout this work, we assume the AEs (and variants thereof) to be symmetric, i.e., the encoder and decoder contain the same number of layers with the same number of neurons but reverse in their order. Furthermore, we assume all variants discussed and used in this paper to apply steady compression, that is, the number of neurons contained in one layer decreases/increases to the next layer by a constant factor. The constant is obtained when knowing the input and latent dimensions, i.e., how much compression should be applied. Doing this removes the need to tune the hidden layer dimensions as additional hyperparameters [9].

For the CVAE, we apply one hidden layer in the encoder and the decoder and a latent dimension of one. We train the CVAE over 50 epochs and apply early stopping with a patience of ten epochs. Furthermore, we sample 64 latent representations per data point. We use the ADAM optimizer [12] for any model training in this work. In the synthetic data experiments we apply a learning rate of 0.001.

We use the implementation by Rothfuss et al. [21] with the default settings. Inference is done as described for the CVAE. In addition, we provide the VAE and KDE as a baseline, which estimates the joint density $p(x_t)$ for all sensors using the train samples x_t. The anomaly score of these models for a sample x_t is then the negative log-likelihood $-log\ p(x_t)$.

For the VAE, we apply two hidden layers in the encoder and the decoder and a latent dimension of two. Furthermore, we sample 64 latent representations per data point. We train the VAE over 100 epochs and apply early stopping with a patience of 10 epochs.

We apply the KDE implementation from scikit-learn [15] with the Epanechnikov kernel and Scotts's rule for the bandwidth selection, which worked best in preliminary experiments. The other parameters are kept default.

Moreover, we report as a benchmark the performance of the conditional density estimate of the variable x_t^i, where the anomalies are injected, given its causal parents $pa(x_t^i)$. We use the negative conditional log-likelihood $-log\ p(x_t^i \mid pa(x_t^i))$ of a variable x_t^i at time t as the anomaly score for inference.

We investigate the impact of the used causal relations. Thus, we apply the causal approach with causal structures that deviate from the ground truth. In the following, we list the considered graph structures:

- We apply the ground truth graph structure.
- We evaluate the fully connected graph for $\tau = 1$, including the autodependencies. This scenario includes, besides all true links, all possible false positives.
- We investigate the graph structure obtained when we apply the PCMCI algorithm to CD in combination with the ParCorr conditional independence test on non-linear data, with $\tau_{min} = 1$, and $\tau_{max} = 2$. This graph structure yields all true links and several false positives (FPs). However, we consider more realistic false positives in this experiment, resulting in lower dimensions for the distributions to estimate than when applying the fully connected graph structure.
- We examine the graph structure consisting of all links at $\tau = 1$ except for the true links. Thus, the used graph structure contains all false negative (FN) and FP links.
- We investigate the graph structure containing no links at all, i.e., we assume no variable is the cause or effect of another.

We report the average precision (AP) in Table 1, and the TPR@{0.05,0.1} in Table 2. The causal approach using CVAEs performs superiorly over the VAE and KDE baselines regarding the AP for all anomaly types. Similar observations can be made regarding the reported TPRs. Furthermore, the individual CVAE of the anomalous variable shows the best AD performance regarding all anomaly types and all reported metrics. Regarding the varying graph structures, we observe the best performance when using the ParCorr graph, which performs better for measurement anomalies than the true graph structure and is on par for intervention anomalies regarding the AP. This indicates the importance of detecting the true causal parents, but also the ability to deal with several false positives (ParCorr graph). However, when the amount of false positives increases, the performance decreases, as can be seen by the performance of the fully connected graph. In addition, when the causal parents are not integrated, as when using no links at all, we obtain performance decreases. Moreover, the true graph structure performs better than all other graph structures in all metrics for effect anomalies and regarding TPR@{0.05,0.1} for measurement and intervention anomalies. We see smaller performance degradations for the differing graph structures for measurement anomalies than for intervention and effect anomalies.

5.2 C-MAPPS

We evaluate the causal approach on the publicly available C-MAPPS dataset [26], which was initially used to predict turbofan engines' Remaining Useful Life (RUL) [8, 26]. The dataset provides multivariate timeseries data from turbofan engine simulations conducted with the Commercial Modular AeroPropulsion System Simulation (C-MAPPS) software [7]. The simulation software can simulate different operation conditions, such as the altitude or the temperature. The dataset comprises four sub-datasets: FD001, FD002, FD003, and FD004. These sub-datasets differ in the simulated operation conditions and the reasons for engine failure. They consist of multiple engine simulations, which can be considered a fleet of engines. Therefore, every dataset contains multiple timeseries, each describing the simulation of one engine. The multivariate timeseries contain 21 measurements/variables of the turbofan engine, such as the fan speed or the temperature at the low-pressure turbine (LPC), for all sensor values, refer to the paper by Saxena et al. [26]. Thus, for every engine, multiple flights are simulated, and the corresponding timeseries reports one measurement value per sensor per flight. The initial wear differs among the engines. We follow Jakubowski

Table 1 Results for AD on the synthetic data with non-linear dependencies. Metric is AP. The best and the second-best results are denoted in bold.

Model	Measurement	Effect	Intervention
CVAE of Anomalous Variable	**0.716**	**0.667**	**0.495**
VAE with x_t	0.434	0.241	0.191
KDE	0.338	0.250	0.215
Causal + CVAE + True Graph	0.621	**0.464**	**0.245**
Causal + CKDE + True Graph	0.470	0.429	0.124
Causal + CVAE + Fully Connected Graph	0.273	0.163	0.190
Causal + CVAE + ParCorr Graph	**0.633**	0.358	0.243
Causal + CKDE + All FPs & FNs	0.604	0.104	0.110
Causal + KDE + No Links	0.514	0.226	0.156

et al. [9] and use the dataset FD004, which they consider the most challenging scenario since six different operation conditions are present and two possible failure modes occur. Training and test sets are provided for every sub-dataset. The train set contains run-to-failure of individual engine simulations. In contrast, the test set's timeseries stop at a point in time before the engines fail. The test set also contains the RUL values of each timeseries, which will be predicted in the original task.

Jakubowski et al. [9] use the dataset for anomaly detection. We follow their definition for labeling the data points as anomalies or normal. Concrete data points with an RUL greater than 130 are assigned to the normal class. Samples with an RUL smaller than 20 are assigned to the anomaly class. The intermediate samples are not considered further. Following the labeling procedure, the test set for RUL prediction yields only a small amount of anomalous samples due to the truncated runs, which often miss the anomalies. Therefore, we use the run-to-failure simulations from the initial train set and assign each run from the C-MAPPS train dataset to the train, the validation, or the test set. In contrast to Jakubowski et al. [9], we do not use labeled validation data to fit a threshold. In Table 3, we depict the number of samples per split. The train and validation splits do not contain anomalous samples since we train unsupervised.

For the causal approach, we first discover the causal relations of the C-MAPPS dataset. Thus, we inspect the joint and marginal densities for the pairs of variables, encountering non-Gaussian densities. In addition, we examine the scatter plots for the pairs of variables, we encounter a strong tendency for linear dependencies. Therefore, we select the RobParCorr test for conditional independence. The analysis of the lagged dependencies, i.e., the unconditional dependencies of the pairs of variables over the various time lags, indicates that the selection of $\tau_{min} = 0$ is adequate, especially since we work with data with a small sample rate, i.e., one measurement per flight. Thus, we apply the PCMCIplus algorithm to be able to obtain

Table 2 Results for AD on the synthetic data with non-linear dependencies. Metric is TPR@FPRs. The best and the second-best results are denoted in bold.

	Measurement		Effect		Intervention	
Model	TPR@0.05	TPR@0.1	TPR@0.05	TPR@0.1	TPR@0.05	TPR@0.1
CVAE of Anomal. Variable	**0.668**	**0.702**	**0.657**	**0.733**	**0.505**	**0.672**
VAE with x_t	0.395	0.565	0.140	0.264	0.148	0.236
KDE	0.313	0.490	0.195	0.346	0.154	0.262
Causal + CVAE + True Graph	**0.626**	**0.678**	**0.458**	**0.581**	0.204	**0.383**
Causal + CKDE + True Graph	0.599	0.643	0.384	0.486	0.079	0.131
Causal + CVAE + Fully Connect. Graph	0.260	0.359	0.110	0.198	0.16	0.242
Causal + CVAE + ParCorr Graph	0.594	0.648	0.359	0.559	**0.233**	0.247
Causal + CKDE + All FPs & FNs	0.585	0.629	0.051	0.116	0.059	0.115
Causal + KDE + No Links	0.548	0.602	0.058	0.385	0.059	0.134

Table 3 The number of samples contained in the datasplits of the C-MAPPS dataset.

	# normal samples	# anomal samples
Train Set	19057	-
Validation Set	5052	-
Test Set	4898	760

also directed contemporaneous links. We select $\tau_{max} = 3$ to integrate the peak in the lagged dependencies at $\tau = 0$ and provide a buffer. We use the implementation from the tigramite package by Runge et al. [22]. Thereby, we do not specify a pc_α value, which is used for the selection of the conditions used in the conditional independence tests, refer to Runge et al. [25, 22]. The remaining parameters of the algorithm are kept default.

Next, we apply the detected graph structure in the causal approach by only using the detected directed links (partial factorization). Thus, we encounter at least one causal parent for every variable. We provide the results for the causal approach based on CVAEs and CKDEs. In addition, we provide a mixture of CVAEs and CKDEs. Thus, we use a CKDE for variables with less than three causal parents and a CVAE for others. For the CVAEs, we keep the hyperparameters as for the synthetic data, except that we evaluate per variable the number of hidden layers, either one or two, and the latent dimension, either one or half of the input dimension of the CVAE (variables dimension plus number of causal parents).

Besides the synthetic data's baselines, we provide the AE as a baseline for the C-MAPPS dataset. Furthermore, we provide the VAE based on the data $(x_t, \ldots, x_{t-3})$. For the VAEs and the AE, we search for the best latent dimension, number of hidden layers, and learning rate in a hyperparameter search. In addition, we search for the number of latent samples for the VAEs.

We report the Area Under the Receiver Operating Characteristic Curve (AUROC), the AP, and the TPR@{0.05,0.1,0.2,0.3} in Table 4. For the VAEs, the causal approach with CVAEs, and the causal approach consisting of CVAEs and CKDEs, we report the mean and the standard error (SE) of the mean of the respective metrics obtained when retraining the best model configuration ten times.

Our results indicate that the VAE outperforms the KDE when considering the joint probability of all variables concerning all reported metrics. Thus, for the VAEs, we encounter superior performance when considering individual observations instead of a row of four observations. Additionally, the SEs are higher when using multiple observations in a row. Moreover, when considering all variables of the timeseries, the AE outperforms the KDE. In contrast, the VAEs show stronger performance than the AE. In the following, we refer with VAE to the variant relying on one observation. The outperformance becomes especially clear when considering the AP, where the VAE achieves with 0.911 a 14 % higher AP than the AE with 0.798. When we look at the results of the causal approach, we encounter worse results for the variant solely relying on the CVAE for the estimation of the conditional distributions than the baselines AE and VAE. For the AP, the causal variant with CVAEs reaches 0.691 with an SE of ca. 0.09, whereas the VAE achieves 0.911 with an SE of approximately 0.004. The VAE reaches for TPRs@$\{0.05, 0.1\}$ with ca. 0.06 as its highest SE, while for the other metrics, the SE is limited to roughly 0.02. The SE of the causal approach using CVAEs is for all metrics between 0.08 and 0.1, which indicates that the CVAE's ability to estimate the distributions can vary strongly. Thus, we develop an ensemble of CVAEs if the conditional dimension is larger than three and CKDEs otherwise. This variant of the causal approach outperforms the causal variants solely relying on CVAEs or CKDEs. In addition, it outperforms the VAE (one observation) in all metrics by approximately 0.01, by 9.3 % regarding the TPR@0.05, and by 10.9 % regarding the TPR@0.1. In addition, the SE of the causal approach with a mixture of CKDEs and CVAEs is approximately 0.

6 Conclusions

In this paper, we leverage causal information to detect anomalies in multivariate timeseries. Therefore, we develop previous work further by discovering the causal relations of a multivariate timeseries using the CD algorithms adapted to the timeseries setting. Next, the causal relations provide the decomposition of the problem into estimating the conditional distribution given its parents for every variable.

Our results from experiments conducted with synthetically generated data indicate that the causal approach performs better than the VAE, relying on the joint probability of the individual observations. In addition, evaluating graph structures that deviate from the ground truth graph structure indicates the importance of detecting the true causal parents, even when detecting some additional false parents. Furthermore, missing out on the true causal parents leads to strong performance degradation. Additionally, we observe that the performance is less dependent on the causal structure for measurement anomalies than for effect and intervention anomalies.

Table 4 The results for AD on the C-MAPPS dataset. The best results are denoted in bold.

Model	AP	AUROC	TPR@0.05	TPR@0.1	TPR@0.2	TPR@0.3
AE with x_t	0.798	0.926	0.720	0.814	0.878	0.917
VAE with x_t	0.911 ± 0.004	0.976 ± 0.011	0.876 ± 0.060	0.888 ± 0.058	0.969 ± 0.015	0.987 ± 0.008
VAE with $(x_t, \ldots, x_{t-3})$	0.859 ± 0.068	0.960 ± 0.023	0.822 ± 0.106	0.867 ± 0.085	0.936 ± 0.046	0.966 ± 0.031
KDE with x_t	0.570	0.711	0.518	0.522	0.542	0.546
Causal + CVAE	0.691 ± 0.086	0.852 ±0.057	0.662 ± 0.096	0.692 ± 0.093	0.754 ± 0.091	0.796 ± 0.082
Causal + CVAE & CKDE	**0.924 ± 0.001**	**0.988 ±0.000**	**0.958 ± 0.001**	**0.985 ± 0.002**	**0.994 ± 0.002**	**0.998 ± 0.000**
Causal + CKDE	0.711	0.955	0.729	0.859	0.963	0.995

Moreover, the experiments on the C-MAPPS dataset show that the causal approach, consisting of an ensemble of CKDEs and CVAEs, performs better than the VAE baselines, which rely on individual observations and rows of observations. We recognize the importance of the individual conditional density estimates being well-fitted since the causal approach, when relying on either one CVAE or CKDEs, performs worse than the mixture.

References

1 Simon D Duque Anton, Sapna Sinha, and Hans Dieter Schotten. Anomaly-based intrusion detection in industrial data with svm and random forests. In *2019 International conference on software, telecommunications and computer networks (SoftCOM)*, pages 1–6. IEEE, 2019.

2 Jianmin Bao, Dong Chen, Fang Wen, Houqiang Li, and Gang Hua. Cvae-gan: fine-grained image generation through asymmetric training. In *Proceedings of the IEEE international conference on computer vision*, pages 2745–2754, 2017.

3 Arpita Bhargava and AS Raghuvanshi. Anomaly detection in wireless sensor networks using s-transform in combination with svm. In *2013 5th International Conference and Computational Intelligence and Communication Networks*, pages 111–116. IEEE, 2013.

4 Enyan Dai and Jie Chen. Graph-augmented normalizing flows for anomaly detection of multiple time series. *arXiv preprint arXiv:2202.07857*, 2022. `arXiv:2202.07857`.

5 Zahra Zamanzadeh Darban, Geoffrey I Webb, Shirui Pan, Charu C Aggarwal, and Mahsa Salehi. Deep learning for time series anomaly detection: A survey. *arXiv preprint arXiv:2211.05244*, 2022. `doi:10.48550/arXiv.2211.05244`.

6 Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35(5), pages 4027–4035, 2021. `doi:10.1609/AAAI.V35I5.16523`.

7 Dean K Frederick, Jonathan A DeCastro, and Jonathan S Litt. User's guide for the commercial modular aero-propulsion system simulation (c-mapss). Technical report, NASA, 2007.

8 Felix O Heimes. Recurrent neural networks for remaining useful life estimation. In *2008 international conference on prognostics and health management*, pages 1–6. IEEE, 2008.

9 Jakub Jakubowski, Przemysław Stanisz, Szymon Bobek, and Grzegorz J Nalepa. Anomaly detection in asset degradation process using variational autoencoder and explanations. *Sensors*, 22(1):291, 2021. `doi:10.3390/S22010291`.

10 Andrew KS Jardine, Daming Lin, and Dragan Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical systems and signal processing*, 20(7):1483–1510, 2006.

11 Mohammed Shaker Kareem and Lamia AbedNoor Muhammed. Anomaly detection in streaming data using isolation forest. In *2024 Seventh International Women in Data Science Conference at Prince Sultan University (WiDS PSU)*, pages 223–228. IEEE, 2024.

12 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

13 Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

14 Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018. doi:10.1109/LRA.2018.2801475.

15 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

16 Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

17 Jeffrey S Racine et al. Nonparametric econometrics: A primer. *Foundations and Trends® in Econometrics*, 3(1):1–88, 2008.

18 Ajay Singh Raghuvanshi, Rajeev Tripathi, and Sudarshan Tiwari. Machine learning approach for anomaly detection in wireless sensor data. *International Journal of Advances in Engineering & Technology*, 1(4):47, 2011.

19 Kishore K Reddy, Soumalya Sarkar, Vivek Venugopalan, and Michael Giering. Anomaly detection and fault disambiguation in large flight data: A multi-modal deep auto-encoder approach. In *Annual conference of the phm society*, volume 8(1), 2016.

20 Ferdinand Rewicki, Joachim Denzler, and Julia Niebling. Is it worth it? comparing six deep and classical methods for unsupervised anomaly detection in time series. *Applied Sciences*, 13(3):1778, 2023.

21 Jonas Rothfuss, Fabio Ferreira, Simon Walther, and Maxim Ulrich. Conditional density estimation with neural networks: Best practices and benchmarks. *arXiv preprint arXiv:1903.00954*, 2019.

22 Jakob Runge. Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets. In *Conference on Uncertainty in Artificial Intelligence*, pages 1388–1397. Pmlr, 2020. URL: http://proceedings.mlr.press/v124/runge20a.html.

23 Jakob Runge, Sebastian Bathiany, Erik Bollt, Gustau Camps-Valls, Dim Coumou, Ethan Deyle, Clark Glymour, Marlene Kretschmer, Miguel D Mahecha, Jordi Muñoz-Marí, et al. Inferring causation from time series in earth system sciences. *Nature communications*, 10(1):2553, 2019.

24 Jakob Runge, Andreas Gerhardus, Gherardo Varando, Veronika Eyring, and Gustau Camps-Valls. Causal inference for time series. *Nature Reviews Earth & Environment*, 4(7):487–505, 2023.

25 Jakob Runge, Peer Nowack, Marlene Kretschmer, Seth Flaxman, and Dino Sejdinovic. Detecting and quantifying causal associations in large nonlinear time series datasets. *Science advances*, 5(11):eaau4996, 2019.

26 Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management*, pages 1–9. IEEE, 2008.

27 Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

28 Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

29 Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2001.

30 Jun-ichi Takeuchi and Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE transactions on Knowledge and Data Engineering*, 18(4):482–492, 2006. doi:10.1109/TKDE.2006.1599387.

31 Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 world wide web conference*, pages 187–196, 2018.

32 Asrul H Yaacob, Ian KT Tan, Su Fong Chien, and Hon Khi Tan. Arima based network anomaly detection. In *2010 Second International Conference on Communication Software and Networks*, pages 205–209. IEEE, 2010.

33 Wenzhuo Yang, Kun Zhang, and Steven CH Hoi. A causal approach to detecting multivariate time-series anomalies and root causes. *arXiv preprint arXiv:2206.15033*, 2022.

34 Yufeng Zhang, Jialu Pan, Li Ken Li, Wanwei Liu, Zhenbang Chen, Xinwang Liu, and Ji Wang. On the properties of kullback-leibler divergence between multivariate gaussian distributions. *Advances in Neural Information Processing Systems*, 36, 2024.

35 Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850. IEEE, 2020. doi:10.1109/ICDM50108.2020.00093.

36 Rui Zhao, Ruqiang Yan, Zhenghua Chen, Kezhi Mao, Peng Wang, and Robert X Gao. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 115:213–237, 2019.

37 Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018.

Minimalist Diagnosis of Discrete-Event Systems

Gianfranco Lamperti
Department of Information Engineering, University of Brescia, Italy

Marina Zanella
Department of Information Engineering, University of Brescia, Italy

Abstract

Model-based diagnosis of discrete-event systems (DESs) is afflicted by two major difficulties, the former being the huge size of the search space, which has a heavy impact on the processing time, the latter being a possibly large number of diagnoses explaining the perceived sequence of observations, which may cause a cognitive overload in human diagnosticians or even delays in post-processing. These difficulties add up and they are exacerbated in critical scenarios where an action must be taken in real-time. To make DES diagnosis viable in these contexts, a Minimalist Diagnosis Engine is presented, which is based on a parsimony principle: instead of computing the set of all diagnoses inherent to the given sequence of observations, only minimal diagnoses are elicited as candidates. Since in this paper, as in most contributions on model-based diagnosis of DESs in the literature, a diagnosis is defined as a set of faults, minimal diagnoses are *subset* minimal. The proposal is justified since minimal diagnoses are suitable for DESs, and since the new diagnosis engine is able to prune the search space, thus reducing the computation effort with respect to a sound and complete method. Moreover, in order to further decrease the execution time, whenever the method is dealing with a new observation, it performs online a (partial) knowledge-compilation so as the portions of the DES space that have already been processed and transformed into chunks of compiled knowledge can speed up the next abductive reasoning steps, relevant to the upcoming observations.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases model-based reasoning, diagnosis during monitoring, discrete-event systems, active systems, subset-minimal diagnosis, dynamical knowledge-compilation, minimalism, laziness

Digital Object Identifier 10.4230/OASIcs.DX.2024.12

Funding We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), Spoke 9 - Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU, specifically by the project *Argumentation for Informed Decisions with Applications to Energy Consumption in Computing* – AIDECC (CUP D53C24000530001).

1 Introduction

Automated diagnosis, which is aimed at finding out what is wrong in a given (natural or synthetic) system, is still a challenge to AI. Model-based diagnosis [9, 25] is a well-founded, principled approach to accomplish this task. Its rationale is to process a model that is specific to the considered system by means of a domain-independent reasoning engine. In 1987, a fundamental work [24] (*a*) defined the concept of *consistency-based* diagnosis, by applying it to static composite systems, and (*b*) adopted *weak* models, i.e. models that describe only the normal behavior of each component.

A consistency-based diagnosis problem instance amounts to an observation that is inconsistent with the given (modeled) system if all the components are assumed to behave normally. Intuitively, a *diagnosis* (result) is a conjecture that some components are behaving abnormally (they are *faulty*) and the rest (i.e. the components that do not belong to the diagnosis) are behaving normally, where such conjecture is consistent with the system description and the observation. Notice that several alternative diagnoses can be found and that, if the models are weak, given a set of components that is a consistency-based diagnosis, any


© Gianfranco Lamperti and Marina Zanella;
licensed under Creative Commons License CC-BY 4.0



superset of it is a consistency-based diagnosis itself. Hence, in consistency-based diagnosis, when weak models are adopted, the set of all the diagnoses can be characterized by the only diagnoses that are *subset-minimal*, where a diagnosis is subset-minimal if no subset of it is a diagnosis. Computing only the minimal diagnoses adheres to the so-called *parsimony principle* [24]. Unfortunately, as explained in [4], this characterization of consistency-based diagnoses does not hold if the adopted models are *strong*, that is, they describe not only the normal behavior of components but also (some) faulty behavioral modes. Analogously, the above characterization does not hold for *abduction-based* diagnosis [23], which necessarily adopts strong models. An abduction-based diagnosis (result) is an assignment of specific modes to the system components such that the conjunction of this assignment with the system description *entails* the (given) observation. The above theoretical distinction between consistency-based and abduction-based diagnosis applies also to model-based diagnosis of dynamical systems, where models have to describe the state changes over time.

This paper deals with model-based diagnosis of dynamical systems represented as discrete-event systems (DESs). The models adopted in this paper are compositional, strong and *complete*, that is, they encompass *all* the normal and abnormal behaviors of the considered components. The complete behavior of each component is represented as a nondeterministic communicating finite automaton endowed with an *initial state*, where each state transition is either normal or affected by a specific *fault* and, orthogonally, either observable, through a specific event, or unobservable. The state evolves over qualitative time, that is, time tags are ignored. The system operation corresponds to a *trajectory*, which is a chronologically ordered sequence of component state-transitions that starts from the initial state of the overall DES (this being the composition of the initial states of its components) and generates a sequence of observable events; each individual event that has been observed is called an *observation* and the whole sequence of the observed events is called a *temporal observation*. The considered DES diagnosis approach is abduction-based as a *diagnosis* is the set of faults on a trajectory that entails the temporal observation perceived so far. Since several trajectories (possibly an unbounded number, if the model allows for unobservable cycles of state transitions) can generate the same given temporal observation, the number of distinct diagnoses, each being a set of faults, may be very large, and a diagnosis may be a superset of another one. Unfortunately, as already remarked, in this context subset-minimal diagnoses do not characterize all the diagnoses (as in fact some trajectories whose relevant set of faults is a superset of a diagnosis may not entail the given sequence of observable events). Hence, most existing approaches to model-based diagnosis of DESs usually produce *all* the diagnoses relevant to the given temporal observation. However, in this paper, where we focus on the task of DES *diagnosis during monitoring*, which issues a new set of diagnoses upon the reception of each new observation, we propose an algorithm that computes minimal diagnoses only. This choice cannot be grounded just on the (legitimate) need to reduce the number of outputted results, each called a *candidate*, since too many candidates may be overwhelming for the human diagnostician who has to make a decision, possibly under stringent time constraints, or even for an artificial real-time agent. In fact, if the intent of our proposal were just reducing the number of candidates, it would be enough to implement a (purposefully efficient) post-processor that could draw the minimal diagnoses from the collection of all diagnoses computed by a sound and complete existing method. Proposing a new diagnostic method is justified only if two conditions hold: minimal diagnoses are suitable for diagnosis of DESs, and the diagnosis method can reduce the computation time with respect to a sound and complete method. The first condition will be discussed in the next section, while Sections 3 and 4 will provide some background and introduce the new diagnosis method, respectively.

The search space of the new method is smaller than that of a sound and complete method, hence also the second condition above is fulfilled. Section 5 hints to previous works in the literature that have some links with this one. A few summarizing remarks and intentions for future research conclude the paper in Section 6.

2 Motivation

We have already distinguished the notion of a *diagnosis*, which is theoretical and defines a domain, from that of a *candidate*, which is an output actually produced by the diagnosis engine at hand. In this paper, as in most contributions on model-based diagnosis of DESs in the literature, a diagnosis is a *set* of faults relevant to a trajectory that entails the given temporal observation. If the method is *sound*, each candidate is a diagnosis; if it is *complete*, each diagnosis is a candidate. Hence, if the method is sound and complete, the set of candidates includes all and only the diagnoses relevant to the considered problem instance. In this section, if a proposition or a definition mentions *all diagnoses*, it implicitly refers to the output produced by a sound and complete DES diagnosis method. Analogously to all contributions on model-based diagnosis of DESs by other authors, this paper assumes that the observations are not affected by any *uncertainty* [12], that is, they are received, without any alteration in their content and number, in their emission order. If the observations are not affected by any uncertainty, then the set of all diagnoses outputted by a sound and complete method includes the *actual diagnosis*, that is, the only diagnosis that reflects what has really happened inside the system. In Section 4 we propose a method that is sound while it is not complete, as it computes subset-minimal diagnoses only. This implies that the actual diagnosis may not be one of the (computed) candidates. However, when several candidates are outputted, rather that analyzing them individually, an interesting piece of information is their intersection. In fact, if (there is no uncertainty in the observations and) all diagnoses are provided (by a sound and complete method), their intersection is a set of faults that have occurred with certainty, as all such faults are included in the actual diagnosis. Luckily, if only the minimal diagnoses are provided, there is no loss in such information, as stated by the following proposition (in this section, the proofs of all propositions are omitted for the sake of space).

▶ **Proposition 1.** *The intersection of all diagnoses equals the intersection of all the subset-minimal diagnoses. If there is no uncertainty in the DES observations, all the faults in such intersection have occurred with certainty.*

Hence, if we compute minimal diagnoses only, we know the same set of faults that have occurred with certainty as if we had computed all diagnoses. This property provides a motivation for computing minimal diagnoses only and plays an important role in diagnosis during monitoring of DESs, as it will be explained in Subsections 2.2 and 2.3.

2.1 Probability

Assuming that faults in a DES are reciprocally independent, following [5], the probability of a candidate is the product of the individual probabilities of each fault in the candidate to occur and each remaining fault not to occur. If the (possibly unknown) probability value is (sensibly assumed to be) less than 0.5 for each fault, be the probabilities of faults equal to each other or different from each other, each minimal diagnosis δ is more probable than any of its supersets $\delta' \supset \delta$. Hence, the (possibly not unique) most probable diagnosis is a minimal diagnosis (although, unfortunately, the remaining minimal diagnoses are not necessarily very probable). Therefore, the set of minimal diagnoses includes (among others) the most probable diagnoses, which is a good point for minimal diagnoses.

2.2 Monotonicity

When the task of diagnosis during monitoring is performed, the diagnostic engine processes each observable event o_i as soon as it has been perceived and produces a new output Δ_i, that is, a new collection of candidates that are inherent to the whole temporal observation $[o_1, \ldots, o_i]$ received so far. If the method is sound and complete, at each monitoring step all the diagnoses relevant to the temporal observation received so far are computed. A property relevant to this task is *monotonicity* [15], which is recalled here below.

▶ **Definition 2** (Monotonicity [15]). *The results $\langle \Delta_1, \ldots, \Delta_n \rangle$ inherent to a DES (diagnosis during) monitoring problem instance are* monotonic *iff $\forall i \in [1..(n-1)]$, we have $\forall \delta_{i+1} \in \Delta_{i+1}, (\exists \delta_i \in \Delta_i, \delta_i \subseteq \delta_{i+1})$.*

Monotonicity holds if not all the candidates in Δ_i are refuted once a new observation o_{i+1} is processed, instead, some of them, possibly extended through the addition of further faults, will become the new candidates in Δ_{i+1}. This property ensures that the faults in the intersection of the candidates produced in a processing step will never be refuted in the subsequent processing steps, as stated in Proposition 3.

▶ **Proposition 3.** *If the results $\langle \Delta_1, \ldots, \Delta_n \rangle$ inherent to a DES (diagnosis during) monitoring problem instance are monotonic, then, $\forall i \in [1..(n-1)]$, the intersection of the candidates in Δ_i is a subset in all the candidates in any Δ_j, $j > i$.*

Notice that Proposition 3 is independent of the method for DES diagnosis during monitoring, provided that it produces monotonic results. It can easily be proven that, if the observations are not affected by any uncertainty, the results generated by a sound method that is either complete or computes minimal diagnoses only are monotonic. This brings to the following proposition.

▶ **Proposition 4.** *Let $\langle \Delta_1, \ldots, \Delta_n \rangle$ be the results inherent to a DES (diagnosis during) monitoring problem where each Δ_i consists of either all diagnoses (produced by a sound and complete method) or minimal diagnoses only. If there is no uncertainty in the observations, the results are monotonic, which in turn implies that the intersection of the candidates in Δ_i, this intersection being a set of faults that have occurred with certainty, is a subset of faults in all the candidates in any Δ_j, $j > i$.*

Hence, when considering minimal diagnoses only, we can identify the same set of faults that have certainly occurred as when we compute all diagnoses, we do not lose any fault identification ability. This is a really good point for minimal diagnoses.

2.3 Diagnosability

Which are the faults that we can identify? This depends on the *diagnosability* of DESs. In order to recall the definition of diagnosability [26], let us introduce some formalism. Let L be the language representing all the trajectories (sequences of state transitions) inherent to a given DES D. Let $L_f \subseteq L$ be the language of all the trajectories of D that include a transition affected by fault f, and $\bar{L_f} \subseteq L_f$ be the language of all the trajectories in L_f where such transition is the last one. Let *Obs* be the function that provides the (chronologically ordered) sequence of observable events that have occurred on a given (chronologically ordered) sequence of state transitions. Let . be the concatenation operator for sequences of state transitions.

▶ **Definition 5** (Diagnosability [26]). *Given a DES D whose set of faults is Σ_f, a fault $f \in \Sigma_f$ is* diagnosable *if* $\forall \tau_1 \in \bar{L_f}, \exists k \in \boldsymbol{N}, \forall \tau_2 : \tau_1.\tau_2 \in L, |Obs(\tau_2)| \geq k \Rightarrow (\forall \tau \in L), (Obs(\tau) = Obs(\tau_1.\tau_2) \Rightarrow (\tau \in L_f))$. *System D is* diagnosable *if all its faults are diagnosable.*

In other words, a fault f is diagnosable if, for whichever sequence of transitions τ_1 that has preceded the occurrence of f, and for whichever sequence of transitions τ_2 (generating a finite number k of observable events) that has followed it, all the trajectories that produce the same temporal observation as $\tau_1.\tau_2$ include the fault. The value k associated (according to Definition 5) with a diagnosable fault is here called the *(diagnosability) delay* of that fault.

▶ **Proposition 6.** *Given a DES D whose set of faults is Σ_f, let $f \in \Sigma_f$ be a* diagnosable *fault with delay $k \in \boldsymbol{N}$. Let $\tau_1.\tau_2$ be the actual trajectory followed by D, where $\tau_1 \in \bar{L_f}$ and $|Obs(\tau_2)| = k$. Let $Obs(\tau_1.\tau_2)$ be the temporal observation (with no uncertainty) provided altogether as input either to a sound and complete method or to a method that computes minimal diagnoses only. Then, all the diagnoses, and minimal diagnoses as well, include fault f, which is therefore included in the intersection of all the diagnoses, as well as in the intersection of all minimal diagnoses.*

What is the impact of diagnosability when the task of diagnosis during monitoring is performed? Proposition 7 provides an answer.

▶ **Proposition 7.** *Given a DES D whose set of faults is Σ_f, let $f \in \Sigma_f$ be a* diagnosable *fault with delay $k \in \boldsymbol{N}$. Let $\langle \Delta_1, \ldots, \Delta_n \rangle$ be the results inherent to a DES (diagnosis during) monitoring problem instance, where every Δ_i consists of either all diagnoses or the minimal diagnoses only. Let us assume that fault f occurs after the reception of the i-th observation and before or upon the reception of the $(i+1)$-th observation. If there is no uncertainty in the observations, fault f belongs to the intersection of the candidates in any Δ_j, $j \geq i + k$, that is, for each step $j \geq i + k$, it belongs to the set of faults that have occurred with certainty.*

Thus, computing minimal diagnoses only (instead of all diagnoses) does not involve any drawback in detecting and isolating diagnosable faults. Unfortunately, there is a drawback if a fault is not diagnosable, that is, if there exists an (ambiguous) trajectory wherein the fault occurs that is indefinitely observationally identical to a trajectory wherein the fault never occurs. If the perceived temporal observation is consistent with an ambiguous trajectory, there will be a pair of diagnoses, one that includes the fault and the other that does not. Hence, the fault will not fall in the intersection of all diagnoses (nor in the intersection of minimal diagnoses), since it is not certain. However, while the set of all diagnoses includes both diagnoses, the set of minimal diagnoses does not necessarily include both of them (and it may include neither of them). Anyway, if we are interested in the set of faults that have certainly occurred rather than in knowing each and every diagnosis, computing minimal diagnoses only does not cause any loss in the identified certain faults nor any longer delay in their identification with respect to computing all diagnoses. This is why we stick to the computation of minimal diagnoses only.

3 Preliminaries

A distributed, asynchronous DES is a network of components that are modeled as finite communicating automata and are connected to other components via *links*. When a DES starts its behavior, each component is in its initial state and links are assumed to be empty. The occurrence of an external event (outside the DES) may trigger a state transition in a component that is sensitive to that event, which may generate internal events towards other

components, resulting in a cascade of state transitions, called a *trajectory* of the DES, which moves the DES from its initial system state to a new system state. The trajectories of a DES $\mathcal{X}$ are confined to a *space*, which is itself a finite automaton, namely $Space(\mathcal{X}) = (\mathbf{T}, X, \tau, x_0)$, where the alphabet $\mathbf{T}$ is the set of component transitions, X is the set of (system) states, where a state is a pair of a tuple of states of components and a tuple of the events within links, τ is the transition function mapping a state and a component transition into a new state, and x_0 is the initial state. Formally, each string (sequence) of component transitions in the language of $Space(\mathcal{X})$ is a trajectory of $\mathcal{X}$.

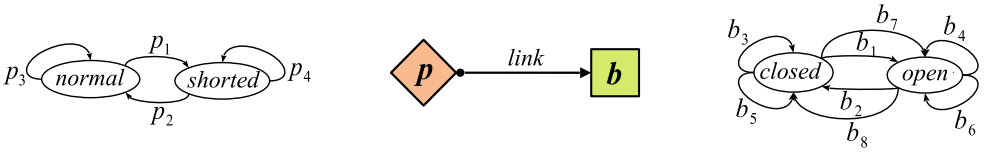


Figure 1 DES watcher $\mathcal{X}_w$ (center), and models of protection p (left) and breaker b (right).

▶ **Example 8.** Outlined in the center of Figure 1 is a DES, called $\mathcal{X}_w$ (*watcher*), which is designed to protect a power transmission line from short circuits. A watcher includes two components, a protection p and a breaker b, and a link from p to b, which serves as a communication channel for the commands sent by the protection to the breaker. When a short circuit strikes a protected line, the protection is expected to command the breaker to open in order to get the short circuit extinguished. If the short circuit is eventually extinguished, the protection commands the breaker to close in order to restore the electric power. The communicating automata (models) of both p (left) and b (right) involve two states, namely *normal* and *shorted* for the protection, and *closed* and *open* for the breaker. Transitions are represented as arrows between states, which are labeled with their identifiers. Component transitions are described in Table 1 (first and second column). Each component transition from a state s to a state s' that is triggered by an input event e and generates a set of output events E is denoted by a triple $\langle s, (e, E), s' \rangle$. If event e is ε, it means that the transition is triggered by an event outside the DES, i.e. an event that is not sent by another component via a link. For instance, transition $p_1 = \langle normal, (\varepsilon, \{op\}), shorted \rangle$ of the protection is triggered by an external event (drop in line voltage, which indicates the occurrence of a short circuit on the line), generates the single event *op* (to open the breaker), and moves the protection from state *normal* to state *shorted*. All transitions of the breaker

Table 1 Description of transition actions for protection p and breaker b in DES $\mathcal{X}_w$ (columns 1 and 2), along with observations and faults in $Map(\mathcal{X}_w)$ (columns 3 and 4).

Transition	*Action*	*Obs*	*Fault*
$p_1 = \langle normal, (\varepsilon, \{op\}), shorted \rangle$	p reacts to a short circuit by generating the open event	**p**	ε
$p_2 = \langle shorted, (\varepsilon, \{cl\}), normal \rangle$	p reacts to a short circuit extinction by generating the close event	**p**	ε
$p_3 = \langle normal, (\varepsilon, \{cl\}), normal \rangle$	p reacts to a short circuit by generating the close event	ε	*pfo*
$p_4 = \langle shorted, (\varepsilon, \{op\}), shorted \rangle$	p reacts to a short circuit extinction by generating the open event	ε	*pfc*
$b_1 = \langle closed, (op, \emptyset), open \rangle$	b reacts to the open event by opening	**b**	ε
$b_2 = \langle open, (cl, \emptyset), closed \rangle$	b reacts to the close event by closing	**b**	ε
$b_3 = \langle closed, (op, \emptyset), closed \rangle$	b reacts to the open event by remaining closed	ε	*bfo*
$b_4 = \langle open, (cl, \emptyset), open \rangle$	b reacts to the close event by remaining open	ε	*bfc*
$b_5 = \langle closed, (cl, \emptyset), closed \rangle$	b reacts to the close event by remaining closed	**b**	ε
$b_6 = \langle open, (op, \emptyset), open \rangle$	b reacts to the open event by remaining open	**b**	ε
$b_7 = \langle closed, (cl, \emptyset), open \rangle$	b reacts to the close event by opening	**b**	*bop*
$b_8 = \langle open, (op, \emptyset), closed \rangle$	b reacts to the open event by closing	**b**	*bcl*

are triggered by an event generated by the protection (either *op* or *cl*), and do not generate output events ($E = \emptyset$). The space of $\mathcal{X}_w$ is depicted in Figure 2, where states are labeled $0 \ldots 7$, with 0 being the initial state. Each state is identified by a pair of states of components p and b, along with the (possibly empty, namely ε) event within the link. For instance, a trajectory of $\mathcal{X}_w$ is $T = [p_3, b_5, p_1, b_3, p_4, b_3, p_2, b_5]$. Due to cycles in the space, $\mathcal{X}_w$ may follow (at least in theory) an infinite number of different trajectories.

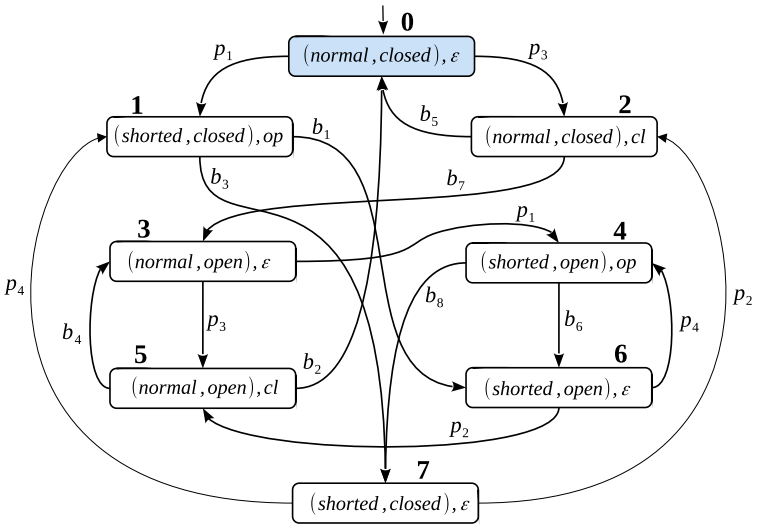


Figure 2 $Space(\mathcal{X}_w)$, the space of the *watcher*, where **0** is the initial state.

To support the diagnosis of a DES $\mathcal{X}$, we need to define both the *observability* and the *abnormality* of $\mathcal{X}$. This is specified by a table, namely $Map(\mathcal{X})$, which is a set of triples (t, o, f), where t is a component transition, o is a (possibly empty, namely ε) observation, and f is a (possibly empty, namely ε) fault. Specifically, if $o \neq \varepsilon$, then t is *observable*, otherwise t is *unobservable*; similarly, if $f \neq \varepsilon$, then t is *faulty*, otherwise t is *normal*. Based on $Map(\mathcal{X})$, each trajectory T of $\mathcal{X}$ is associated with a *temporal observation*, which is the sequence of the observations associated with the observable component transitions in T, namely $Obs(T) = [\, o \mid t \in T, (t, o, f) \in Map(\mathcal{X}), o \neq \varepsilon \,]$. A trajectory T is said to *conform* with a temporal observation $\mathcal{O}$ iff $Obs(T) = \mathcal{O}$. $Map(\mathcal{X})$ also associates T with a *diagnosis*, which is the set of faults associated with the component transitions in T, namely $Dgn(T) = \{\, f \mid t \in T, (t, o, f) \in Map(\mathcal{X}), f \neq \varepsilon \,\}$. The *diagnosis set* $\mathcal{D}$ of $\mathcal{O}$ is the set of diagnoses of the trajectories of $\mathcal{X}$ conforming with $\mathcal{O}$ (i.e. it is the set of *all diagnoses*, according to the terminology in Section 2), namely: $\mathcal{D}(\mathcal{O}) = \{Dgn(T) \mid T \in Space(\mathcal{X}), \mathcal{O} = Obs(T)\}$. Let $\mathcal{O}_i$ denote a nonempty prefix $[o_1, \ldots, o_i]$ of $\mathcal{O}$, $i \geq 1$. The *temporal diagnosis set* $\mathbf{D}$ of $\mathcal{O}$ is the sequence of diagnosis sets $\mathcal{D}(\mathcal{O}_i)$, $i \geq 1$, namely: $\mathbf{D}(\mathcal{O}) = [\mathcal{D}(\mathcal{O}_1), \mathcal{D}(\mathcal{O}_2), \ldots]$.

▶ **Example 9.** For DES $\mathcal{X}_w$, we assume two possible observations, namely **p**: the protection performs a normal action, and **b**: the breaker performs a (possibly faulty) action. We also assume six possible faults, namely *pfo*: the protection fails to send the open command, *pfc*: the protection fails to send the close command, *bfo*: the breaker fails to open, *bfc*: the breaker fails to close, *bop*: the breaker opens instead of closing, and *bcl*: the breaker closes instead of opening. Based on these sets of observations and faults, the observability and abnormality of $\mathcal{X}_w$ is defined in $Map(\mathcal{X}_w)$, which is embedded in Table 1 (third and fourth columns), where both an observation label and a fault label are associated with each component transition.

Accordingly, transition p_1 is observable and normal, p_3 is unobservable and faulty, while b_7 is both observable and faulty. Since the same observation is associated with several transitions, the component transition that actually occurred cannot be univocally identified based on the given observation only. With trajectory $T = [p_3, b_5, p_1, b_3, p_4, b_3, p_2, b_5]$, we have $Obs(T) = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}]$ and $Dgn(T) = \{pfo, pfc, bfo\}$. Based on $Space(\mathcal{X}_w)$, we can find that the diagnosis set of $Obs(T)$ includes six diagnoses: $\{pfo, bfo\}$, $\{pfo, pfc, bfo\}$, $\{pfo, bfo, bop\}$, $\{pfo, pfc, bfo, bop\}$, $\{pfo, bfo, bfc, bop\}$, and $\{pfo, pfc, bfo, bfc, bop\}$. Even if the diagnosis set involves the actual diagnosis $\{pfo, pfc, bfo\}$ relevant to trajectory T, due to ambiguity, five additional diagnoses are embodied in that set. The temporal diagnosis set of $\mathcal{O}$ is $[\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4]$, where $\mathcal{D}_1 = \{\{pfo\},\{pfo, bop\},\{pfo, bfc, bop\}\}$, $\mathcal{D}_2 = \{\{pfo\}, \{pfo, bfo\}, \{pfo, bop\}, \{pfo, pfc, bfo\}, \{pfo, bfc, bop\}\}$, $\mathcal{D}_3 = \{\{pfo, bfo\}, \{pfo, pfc, bfo\}, \{pfo, bfo, bop\}, \{pfo, pfc, bfo, bfc\}, \{pfo, pfc, bfo, bop\}, \{pfo, bfo, bfc, bop\}, \{pfo, pfc, bfo, bfc, bop\}\}$, and $\mathcal{D}_4 = \{\{pfo, bfo\}, \{pfo, pfc, bfo\}, \{pfo, bfo, bop\}, \{pfo, pfc, bfo, bop\}, \{pfo, bfo, bfc, bop\}, \{pfo, pfc, bfo, bfc, bop\}\}$. We can easily appreciate that the results (in the temporal diagnosis set) are monotonic (cf. Definition 2), as expected based on Proposition 4. The sequence of the intersections of all candidates in each $\mathcal{D}_i$, $i \in [1\,..\,4]$, is $[\{pfo\}, \{pfo\}, \{pfo, bfo\}, \{pfo, bfo\}]$, where each intersection provides a set of faults that have occurred with certainty. Thus, after processing the first observation, we know that fault *pfo* has certainly occurred, the same as after the second observation. Notice that each intersection for $i \in [1\,..\,2]$ equals the actual diagnosis (relevant to $\mathcal{O}_i$). After processing the third observation, we know that also fault *bfo* has certainly occurred. The fourth observation does not add any new certain fault. The intersections for $i \in [3\,..\,4]$ do not involve fault *pfc*, although it has actually occurred in T. Given $Map(\mathcal{X}_w)$, fault *pfc* is not diagnosable. A proof of its non-diagnosability is provided, for instance, by two trajectories, $[p_3, b_5, p_1, b_3, p_4]$ and $[p_3, b_5, p_1]$, which are observationally identical and end in the same state (1). The former trajectory is affected by fault *pfc* (associated with p_4) while the latter is not. Both trajectories can go on indefinitely, producing the same observations, and, if they will not follow p_4 any more, the occurrence of *pfc* will be uncertain for ever. For instance, if the next transitions of both trajectories are $[b_3, p_2, b_5]$, the former is indeed T, which is why fault *pfc* does not belong to the intersections for $i \in [3\,..\,4]$.

4 Minimalist diagnosis

We now present a Minimalist Diagnosis Engine for DESs, which generates the set of minimal diagnoses at the reception of each newly-occurred observation with the support of a minimalist diagnoser generated lazily. The algorithm makes use of the definitions given below.

▶ **Definition 10.** *Let $\mathcal{D}$ be the diagnosis set of a temporal observation $\mathcal{O}$. A diagnosis $\delta \in \mathcal{D}$ is* minimal *iff there is no other diagnosis $\delta' \in \mathcal{D}$ such that $\delta' \subset \delta$. The* candidate set *of $\mathcal{O}$, $\Delta(\mathcal{O})$, is the set of minimal diagnoses in $\mathcal{D}$.*

▶ **Example 11.** With reference to Example 9, the candidate set of the temporal observation $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}]$ is the singleton $\Delta(\mathcal{O}) = \{\{pfo, bfo\}\}$, as every other diagnosis in the diagnosis set of $\mathcal{O}$ is a superset of $\{pfo, bfo\}$.

▶ **Definition 12.** *Let $\mathcal{O} = [o_1, o_2, \ldots]$ be a temporal observation of $\mathcal{X}$. The* temporal candidate set *of $\mathcal{O}$ is the sequence of candidate sets $\Delta(\mathcal{O}_i)$, $i \geq 1$, namely, $\boldsymbol{\Delta}(\mathcal{O}) = [\Delta(\mathcal{O}_1), \Delta(\mathcal{O}_2), \ldots]$, where $\mathcal{O}_i$ is the prefix of $\mathcal{O}$ up to the i-th observation.*

▶ **Example 13.** With reference to the temporal diagnosis set of $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}]$ in Example 9, the temporal candidate set of $\mathcal{O}$ is $\mathbf{\Delta}(\mathcal{O}) = [\{\{pfo\}\}, \{\{pfo\}\}, \{\{pfo, bfo\}\}, \{\{pfo, bfo\}\}]$, where, incidentally and in contrast with the temporal *diagnosis* set, each candidate set is a singleton. We can easily appreciate that also these results, relevant to minimal diagnoses only, are monotonic, as stated by Proposition 4. The sequence of the intersections of all candidates in each Δ_i, $i \in [1\,..\,4]$, is $[\{pfo\}, \{pfo\}, \{pfo, bfo\}, \{pfo, bfo\}]$, where each intersection consists of faults that have occurred with certainty. As expected, this sequence is the same as when all candidates are computed (cf. Example 9).

▶ **Definition 14.** *An* abduction item *of a DES $\mathcal{X}$ is a pair $\Im = (x, \delta)$, where x is a state in $Space(\mathcal{X})$ and δ is a diagnosis of a trajectory of $\mathcal{X}$ ending in x. The* frontier *of $\Im$, $Front(\Im)$, is a set of triples (x', f, o) where $\langle x, t, x' \rangle$ is a transition in $Space(\mathcal{X})$ and $(t, o, f) \in Map(\mathcal{X})$. A* candidate item *of $\mathcal{X}$ is a triple (x, δ, o), where (x, δ) is an abduction item of $\mathcal{X}$ and $o \neq \varepsilon$ is an observation such that $(t, o, f) \in Map(\mathcal{X})$ and t is the last transition of a trajectory of $\mathcal{X}$ ending in x.*

▶ **Example 15.** With reference to $Space(\mathcal{X}_w)$ in Figure 2 and $Map(\mathcal{X}_w)$ in Example 9, an abduction item of $\mathcal{X}_w$ is $\Im = (4, \{pfo, bop\})$, where $4 = ((shorted, open), op)$ is a state of $\mathcal{X}_w$ and $\{pfo, bop\}$ is the diagnosis of trajectory $[p_3, b_7, p_1]$ of $\mathcal{X}_w$ ending in state 4. The frontier of $\Im$ is the set $Front(\Im) = \{(6, \mathbf{b}, \varepsilon), (7, \mathbf{b}, bcl)\}$. A candidate item of $\mathcal{X}_w$ is $(7, \{pfo, bop, bcl\}), \mathbf{b})$, where $7 = ((shorted, closed), \varepsilon)$ is a state of $\mathcal{X}_w$ and $\{pfo, bop, bcl\}$ is the diagnosis of trajectory $[p_3, b_7, p_1, b_8]$ of $\mathcal{X}_w$ ending in 7, with $(b_8, \mathbf{b}, bcl) \in Map(\mathcal{X}_w)$.

▶ **Definition 16.** *Let $\mathbf{O}$ be the domain of observations of a DES $\mathcal{X}$ with initial state x_0. The* minimalist diagnoser *of $\mathcal{X}$ is a finite automaton*

$$Mind(\mathcal{X}) = (\mathbf{O}, D, \tau, d_0) \tag{1}$$

where D is the set of states $(\mathbb{I}, \mathbb{D})$, with $\mathbb{I}$ being a set of abduction items of $\mathcal{X}$ and $\mathbb{D}$ a set of diagnoses, $d_0 = (\{(x_0, \emptyset)\}, \{\emptyset\})$ is the initial state, and $\tau : D \times \mathbf{O} \mapsto D$ is the transition function, where $\tau((\mathbb{I}, \mathbb{D}), o) = (\mathbb{I}', \mathbb{D}')$ iff, for each $(x, \delta) \in \mathbb{I}$, there is in $Space(\mathcal{X})$ a sequence $[t_1, \ldots, t_k]$ of contiguous transitions, $k \geq 1$, where t_1 exits state x, all transitions $t_1, \ldots, t_{k-1}$ are unobservable while t_k is observable via observation o and enters a state x', δ' is the extension of δ by the faults involved in transitions $t_1, \ldots, t_k$, $(x', \delta') \in \mathbb{I}'$ provided that there is no other abduction item $(x', \bar{\delta}) \in \mathbb{I}'$ such that $\bar{\delta} \subset \delta'$, and $\mathbb{D}'$ is the minimal set of diagnoses in $\{\delta' \mid (x', \delta') \in \mathbb{I}'\}$.

▶ **Example 17.** A portion of the minimalist diagnoser of DES $\mathcal{X}_w$ is shown in Figure 5. The upper and lower parts of each of the eight depicted states $d_0 \ldots d_7$ display the relevant set $\mathbb{I}$ of abduction items and the set $\mathbb{D}$ of minimal diagnoses, respectively.

▶ **Proposition 18.** *Let $\mathcal{O} = [o_1, \ldots, o_n]$, $n \geq 0$, be a temporal observation of a DES $\mathcal{X}$. Let $d = (\mathbb{I}, \mathbb{D})$ be a state of $Mind(\mathcal{X}) = (\mathbf{O}, D, \tau, d_0)$ such that, if $n = 0$, then $d = d_0$, else $d_1 = \tau(d_0, o_1)$, $d_2 = \tau(d_1, o_2)$, …, $d_n = \tau(d_{n-1}, o_n)$. $\mathbb{D}$ equals the candidate set $\Delta(\mathcal{O})$.*

Proof. By induction on $\mathcal{O}$.

(*Basis*). *The property holds for $\mathcal{O}_0 = [\,]$.* In fact, in this case, $d = d_0 = (\{(x_0, \emptyset)\}, \{\emptyset\})$, where $\mathbb{D} = \{\emptyset\}$. Based on Definition 10, the candidate set $\Delta(\mathcal{O}_0)$ is the set of minimal diagnoses in the diagnosis set $\mathcal{D}(\mathcal{O}_0) = \{Dgn(T) \mid T \in Space(\mathcal{X}), \mathcal{O}_0 = Obs(T)\}$. Since $\mathcal{O}_0$ is empty, the diagnosis relevant to the empty trajectory belongs to $\mathcal{D}(\mathcal{O}_0)$. Since the diagnosis of the empty trajectory is $\emptyset$, the empty diagnosis cannot be a superset of any other diagnosis in $\mathcal{D}(\mathcal{O}_0)$, and every other diagnosis in $\mathcal{D}(\mathcal{O}_0)$ is a superset of it. Hence, $\Delta(\mathcal{O}_0) = \mathbb{D} = \{\emptyset\}$.

(*Induction*). *If the property holds for* $\mathcal{O}_i = [o_1, \ldots, o_i]$, $0 \leq i < n$, *then it also holds for* $\mathcal{O}_{i+1} = [o_1, \ldots, o_{i+1}]$. Considering state $d_i = (\mathbb{I}_i, \mathbb{D}_i)$, based on Definition 16, for each trajectory T of $\mathcal{X}$ conforming with $\mathcal{O}_i$, $\mathbb{I}_i$ includes all abduction items (x, δ) such that x is the state reached by T, and δ is a minimal diagnosis in $\{Dgn(T') \mid T' \in Space(\mathcal{X}), T' = Obs(\mathcal{O}), T' \text{ ending in } x\}$. This will not jeopardize the completeness of candidate sets because, given two abduction items (x, δ) and (x, δ') where $\delta' \supset \delta$, all minimal diagnoses generated by trajectories exiting x and generating δ' in x can also be generated by the trajectories exiting x and generating δ in x, since the conformity of a trajectory with the rest of the temporal observation depends only on state x, not on the associated diagnosis in the abduction item. On the other hand, in order to preserve completeness, it is necessary keeping both abduction items (x, δ) and (x', δ') in $\mathbb{I}_i$ where $x \neq x'$ and $\delta' \supset \delta$, since we do not know whether the trajectories exiting x will still conform with the rest of the temporal observation. Thus, the set $\mathbb{D}'_i = \{\delta \mid (x, \delta) \in \mathbb{I}_i\}$ will include all the diagnoses in candidate set $\Delta(\mathcal{O}_i)$. This is also (trivially) true for $\mathbb{D}'_0$. Moreover, for the same reasons, based on Definition 16, $\mathbb{I}_{i+1}$ is such that $\mathbb{D}'_{i+1} = \{\delta \mid (x, \delta) \in \mathbb{I}_{i+1}\}$ includes all the diagnoses in $\Delta(\mathcal{O}_{i+1})$. Hence, $\mathbb{D}_{i+1} = \Delta(\mathcal{O}_{i+1})$. ◀

▶ **Corollary 19.** *Let* $\mathcal{O} = [o_1, \ldots, o_n]$, $n \geq 0$, *be a temporal observation of a DES* $\mathcal{X}$. *Let* $[d_1, \ldots, d_n]$ *be the sequence of states in minimalist diagnoser* $Mind(\mathcal{X}) = (\mathbf{O}, D, \tau, d_0)$, *where, for each state* $d_i = (\mathbb{I}_i, \mathbb{D}_i)$, $i \in [1\,..\,n]$, $d_1 = \tau(d_0, o_1), d_2 = \tau(d_1, o_2), \ldots, d_i = \tau(d_{i-1}, o_i)$. *The sequence* $[\mathbb{D}_1, \ldots, \mathbb{D}_n]$ *equals the temporal candidate set* $\boldsymbol{\Delta}(\mathcal{O})$.

4.1 Minimalist diagnosis engine

The Minimalist Diagnosis Engine algorithm (lines 1–38) takes as input a temporal observation $\mathcal{O} = [o_1, o_2, \ldots]$ of a DES $\mathcal{X}$ (with initial state x_0), and generates as output a sequence $\boldsymbol{\Delta}$, which is in fact the temporal candidate set of $\mathcal{O}$. The algorithm exploits four main internal data structures: a set $\mathcal{G}$ of abduction items *generated* already, a stack $\mathcal{U}$ of abduction items *under* processing, that is, relevant to the current observation, a set $\mathcal{C}$ of *candidate* items (x, δ, o') relevant to any possible next observation o', and the minimalist diagnoser of $\mathcal{X}$, namely $\mathcal{M} = Mind(\mathcal{X})$. Note that $\mathcal{M}$ is not constructed upfront (offline), but only generated lazily based on the sequence of observations in $\mathcal{O}$. In other words, at any point of the processing, $\mathcal{M}$ represents the part of $Mind(\mathcal{X})$ materialized so far (online) by the algorithm. In line 1, $\mathcal{G}$ and $\mathcal{U}$ are initialized with the abduction item $\Im_0 = (x_0, \emptyset)$. In line 2, the initial state d_0 of $\mathcal{M}$ is generated (as unmarked) and assigned to d, which represents the state of $\mathcal{M}$ relevant to the matching of the current prefix of $\mathcal{O}$ (initially, this is the empty sequence). The aim is to generate the transition function of d (if not materialized already, that is, when d is unmarked), so that, at the next occurring observation o', the relevant candidate set is already computed and stored in the target state d' of $\mathcal{M}$, based on transition $\langle d, o', d' \rangle$. After the initialization of the data structures, the rest of the algorithm consists of two nested loops. The main loop (lines 3–38) is repeated until the DES stops being operated (no further observation can be generated). The nested loop (lines 5–22) is repeated until stack $\mathcal{U}$ becomes empty, in which case no further new abduction item can be generated for the current observation. The idea is to keep generating the frontier of the abduction items in stack $\mathcal{U}$ up to any possible next observation, based on the corresponding trajectories of $\mathcal{X}$, thereby possibly updating δ in each successive abduction item. In the first statement of the main loop (line 4), if the current state d of $\mathcal{M}$ is not marked (that is, if the transition function of d has not been materialized already), the nested loop is executed. At each iteration, an abduction item $\Im = (x, \delta)$ is popped from $\mathcal{U}$ (line 6) and, if it is unmarked in $\mathcal{G}$, that is,

Algorithm 1 Minimalist Diagnosis Engine.

input : $\mathcal{O} = [o_1, o_2, \ldots]$, a temporal observation of a DES $\mathcal{X}$ having initial state x_0
output : $\mathbf{\Delta}$, the temporal candidate set of $\mathcal{O}$

$\Im_0 \leftarrow (x_0, \emptyset)$, $\mathcal{G} \leftarrow \{\Im_0\}$, $\mathcal{U} \leftarrow \lfloor \Im_0 \rfloor$, $\mathcal{C} \leftarrow \emptyset$, $\mathbf{\Delta} \leftarrow [\,]$
Create the initial state $d_0 = (\{\Im_0\}, \{\emptyset\})$ of the minimalist diagnoser $\mathcal{M}$ of $\mathcal{X}$, $d \leftarrow d_0$
repeat
  if *the current state d of $\mathcal{M}$ is not marked* **then**
    repeat
      Pop an abduction item $\Im = (x, \delta)$ from $\mathcal{U}$
      if $\Im$ *is unmarked in* $\mathcal{G}$ **then**
        foreach $(x', f, o) \in \mathit{Front}(\Im)$ **do**
          $\delta' \leftarrow \delta \uplus f$, $\Im' \leftarrow (x', \delta')$
          if $o = \varepsilon$ **then**
            if $\Im' \notin \mathcal{G}$ **then**
              if $(x', \bar{\delta}) \in \mathcal{G}$ *where* $\bar{\delta} \subseteq \delta'$ **then**
                Mark $\Im'$
              else if $x' \neq x$ **then**
                Mark in $\mathcal{G}$ every $(x', \bar{\delta})$ where $\bar{\delta} \supset \delta'$
                Push the abduction item $\Im'$ onto $\mathcal{U}$
              Insert (the possibly marked) abduction item $\Im'$ into $\mathcal{G}$
            else if *there is no* $(x', \bar{\delta}, o) \in \mathcal{C}$ *where* $\delta' \supseteq \bar{\delta}$ **then**
              Insert (x', δ', o) into $\mathcal{C}$
              Remove from $\mathcal{C}$ every (x', δ'', o) where $\delta'' \supset \delta'$
        Mark $\Im$ in $\mathcal{G}$
    until $\mathcal{U}$ *is empty*
    Let $\mathbf{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be a partition of $\mathcal{C}$, where each $\mathcal{C}_j$, $j \in [1\,..\,k]$, includes the candidate items (x, δ, o'_j) relevant to observation o'_j
    foreach $\mathcal{C}_j \in \mathbf{C}$, $\mathbb{I}_j = \{(x, \delta) \mid (x, \delta, o'_j) \in \mathcal{C}_j\}$ **do**
      if $d_j = (\mathbb{I}_j, _)$ *is not a state already generated in* $\mathcal{M}$ **then**
        $\mathbb{D}_j \leftarrow \emptyset$
        foreach *abduction item* $(x', \delta') \in \mathbb{I}_j$ **do**
          if *there is no diagnosis* $\bar{\delta} \in \mathbb{D}_j$ *where* $\bar{\delta} \subseteq \delta'$ **then**
            Insert the diagnosis δ' into $\mathbb{D}_j$
            Remove from $\mathbb{D}_j$ every diagnosis δ'' where $\delta'' \supset \delta'$
        Generate a state $d_j = (\mathbb{I}_j, \mathbb{D}_j)$ in $\mathcal{M}$
      Generate a transition $\langle d, o'_j, d_j \rangle$ in $\mathcal{M}$
    Mark d
  $\mathcal{G} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$
  Let o' be the next observation in $\mathcal{O}$, $\langle d, o', d' \rangle$ a transition in $\mathcal{M}$, where $d' = (\mathbb{I}', \mathbb{D}')$
  Append $\mathbb{D}'$ to $\mathbf{\Delta}$
  $d \leftarrow d'$, $\mathcal{G} \leftarrow \mathbb{I}'$, $\mathcal{U} \leftarrow \mathbb{I}'$
until $\mathcal{X}$ *stops being operated.*

neither processed nor pruned, each triple (x', f, o) in its frontier is considered (lines 8–20). First, an abduction item $\Im' = (x', \delta')$ is generated (line 9), where $\delta' = \delta \uplus f$ is the extension of δ by f, which has no effect when $f = \varepsilon$. Then, two scenarios are considered: either when $o = \varepsilon$ (lines 10–17) or $o \neq \varepsilon$ (lines 18–20). When $o = \varepsilon$, abduction item $\Im'$ is still relevant to the current observation. Thus, if not already processed (line 11), in order to avoid processing abduction items derived from $\Im'$ that cannot lead to new minimal diagnoses, $\Im'$ is marked in case δ' is a superset of a diagnosis $\bar{\delta}$ relevant to an abduction item sharing the same state x' in $\Im'$ (lines 12–13); otherwise (lines 14–16), if $x' \neq x$, every abduction item $(x', \bar{\delta})$, where $\bar{\delta}$ is a (strict) superset of δ', is marked to avoid computing non-minimal diagnoses: in this case, $\Im'$ is pushed onto $\mathcal{U}$ (line 16). In either case, $\Im'$ (which may have been marked in line 13) is eventually inserted into $\mathcal{G}$ (line 17). When, instead, o is observable (lines 18–20), a new candidate item (x', δ', o) is inserted into $\mathcal{C}$ provided that there is no other candidate item $(x', \bar{\delta}, o)$ in $\mathcal{C}$ (that is, relevant to same state x' and same observation o) where δ' is a superset of $\bar{\delta}$ (otherwise, candidate item (x', δ', o) is bound to lead to a non-minimal diagnosis when processing the next observation o). Furthermore, once inserted the new candidate item, all the other candidate items (x', δ'', o) in $\mathcal{C}$, where δ'' is a (strict) superset of δ', are removed for the same reason (non-minimality). At the end of the iteration (line 21), once all triples in the frontier of $\Im$ have been processed (and the corresponding new abduction items $\Im'$ have been generated), abduction item $\Im$ is marked in $\mathcal{G}$. When $\mathcal{U}$ becomes empty, the nested loop terminates (line 22). Now, the candidate set $\mathcal{C}$ contains the initial abduction items for every possible next observation $o'_1, \ldots, o'_k$, which makes it possible to materialize the transition function of the current state d of $\mathcal{M}$. To this end, a partition $\mathbf{C}$ of $\mathcal{C}$ is considered based on the possible next observations $o'_1, \ldots, o'_k$, so that the projection of each $\mathcal{C}_j \in \mathbf{C}$, $j \in [1\,..\,k]$, on the first two elements of the candidate items, namely x and δ, is in fact the set of abduction items identifying state d_j, where $\langle d, o'_j, d_j \rangle$ is a transition in $\mathcal{M}$ to be materialized. This is why, in the loop in lines 24–32, the set $\mathbb{I}_j$ of abduction items (relevant to observation o'_j) is determined in order to possibly generate (if not already generated) the target state d_j in $\mathcal{M}$. In lines 26–31, the corresponding candidate set $\mathbb{D}_j$ is computed. Specifically, in lines 27–30, for each abduction item (x', δ') in $\mathbb{I}_j$, δ' is inserted into $\mathbb{D}_j$ provided that, in order to preserve minimality, $\mathbb{D}_j$ does not include any diagnosis $\bar{\delta}$ that is a subset of δ'. Moreover, if δ' is inserted into $\mathbb{D}_j$, every diagnosis in $\mathbb{D}_j$ that is a superset of δ' is removed (line 30). Once $\mathbb{D}_j$ is computed, a new state $d_j = (\mathbb{I}_j, \mathbb{D}_j)$ is generated in $\mathcal{M}$ (line 31). Eventually, in line 32, a new transition $\langle d, o'_j, d_j \rangle$ is created in $\mathcal{M}$, which allows for the immediate generation of the candidate set $\mathbb{D}_j$ in case the next observation will be o'_j. Once all transitions exiting d have been materialized, the current state d of $\mathcal{M}$ is marked (line 33), meaning that the materialization of the transition function of d has been completed. At this point, in order to process the next occurring observation, the sets $\mathcal{G}$ and $\mathcal{C}$ are emptied (line 34). When a new observation o' occurs (line 35), $\mathcal{M}$ surely includes the relevant transition $\langle d, o', d' \rangle$, where $d' = (\mathbb{I}', \mathbb{D}')$, with $\mathbb{D}'$ being the next diagnosis (candidate) set, which is then appended to $\boldsymbol{\Delta}$ without any additional processing. Before ending the iteration of the main loop, the current state d of $\mathcal{M}$ is changed to d' (the state reached by d with observation o'), while set $\mathcal{G}$ is set to $\mathbb{I}'$, the new initial abduction items, which are also pushed into stack $\mathcal{U}$ (symbol "$\twoheadleftarrow$" denotes this collective push operation): this way, the nested loop will start its computation with the initial abduction items relevant to the newly-occurred observation o'.

▶ **Example 20.** Consider the temporal observation for the DES given in Example 9, which has been extended by a further observation $\mathbf{b}$, namely $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}, \mathbf{b}]$. To trace the run of Minimalist Diagnosis Engine on $\mathcal{O}$, shown in Figure 3 is the space of $\mathcal{X}_w$ (cf. Figure 2) where each component transition identifier t has been replaced by the observation and fault

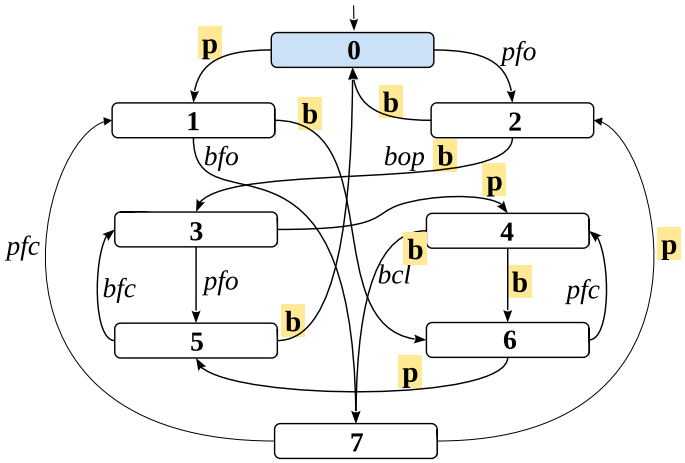


Figure 3 $Space(\mathcal{X}_w)$, with each component transition being replaced by the corresponding observation and fault defined in $Map(\mathcal{X}_w)$ (Table 1), while space states are identified by numbers.

associated with t in $Map(\mathcal{X}_w)$. For instance, transition b_7 marking the arc from state 2 to state 3 in $Space(\mathcal{X}_w)$ is substituted by labels **b** and *bop*, as $(b_7, \mathbf{b}, bop) \in Map(\mathcal{X}_w)$. It should be plain, however, that the labeling of transitions in $Space(\mathcal{X}_w)$ with observations and faults is only instrumental for the clarity of the example: it is not necessary to the algorithm. Or better still, $Space(\mathcal{X}_w)$ is neither available nor necessary to the engine. The generation of the abduction and candidate items is outlined in Figure 4, where numbers $1\,..\,5$ on the left indicate the iterations of the main loop, while the labels on the right are the observations involved in $\mathcal{O}$. The graphs displayed in the center represent the genesis of the abduction items (ellipses), namely (x, δ), and the candidate items in $\mathcal{C}$ (rectangles), namely (x, δ, o). Specifically, an arc from a node $\Im$ (an abduction item) to a node α (either an abduction item $\Im'$ or a candidate item), which is possibly marked with an observation o and/or a fault f, indicates that α is somewhat derived from a triple $(x', f, o) \in Front(\alpha)$ (line 8). The figures of each iteration are listed in Table 2, specifically, the initial instance of stack $\mathcal{U}$, the instance of sets $\mathcal{G}$ and $\mathcal{C}$ at the end of the nested loop (line 22), the newly-occurred observation o' (line 35), and the instance of the set $\mathbb{D}'$ of state d' (line 35), which is in fact the new candidate set appended to $\mathbf{\Delta}$ in line 36. Furthermore, depicted in Figure 5 is the part of minimalist diagnoser $Mind(\mathcal{X}_w)$ materialized in $\mathcal{M}$ by the algorithm based on $\mathcal{O}$. Details of each iteration are given below.

Iteration 1

Initially, the only item in $\mathcal{U}$ is $\Im_0 = (0, \emptyset)$, which is in fact popped from $\mathcal{U}$ in line 6 at the first iteration of the main loop. According to Figure 3, the frontier of $\Im_0$ includes two triples: $(1, \varepsilon, \mathbf{p})$ and $(2, pfo, \varepsilon)$. Since the former involves observation **p**, there is an arc from abduction item $(0, \emptyset)$ to candidate item $(1, \emptyset, \mathbf{p})$, which is inserted into $\mathcal{C}$ (line 19). The latter, instead, leads to a new abduction item $\Im' = (2, \{pfo\})$, which is pushed onto $\mathcal{U}$ (line 16). Note that each arc is marked with the same labels marking the corresponding arc in Figure 3. The processing of the new abduction item $(2, \{pfo\})$ leads to the creation of candidate items $(0, \{pfo\}, \mathbf{b})$ and $(3, \{pfo, bop\}, \mathbf{b})$, which are inserted into $\mathcal{C}$. The relevant instances of $\mathcal{U}$, $\mathcal{G}$, and $\mathcal{C}$ pertaining to the first iteration are listed in the first line of Table 2. At the end of the nested loop (line 22), the transition function of state d_0 is generated based on the partition $\mathbf{C}$ of $\mathcal{C}$, which leads to the creation of the new states d_1 and d_2 in $\mathcal{M}$, along with

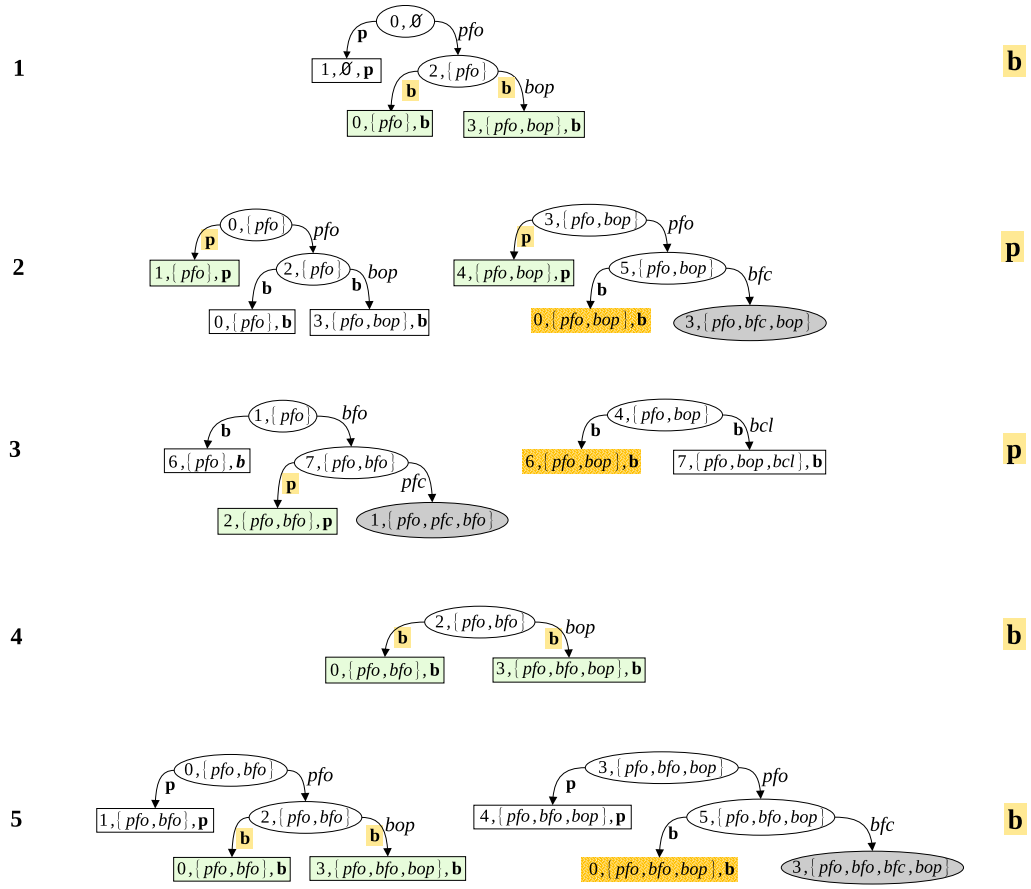


Figure 4 Trace of Minimalist Diagnosis Engine applied to $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}, \mathbf{b}]$ of DES $\mathcal{X}_w$.

their entering transitions $\langle d_0, \mathbf{p}, d_1 \rangle$ and $\langle d_0, \mathbf{b}, d_2 \rangle$. Note how field $\mathbb{D}$ in state d_2 includes only the minimal diagnosis $\{pfo\}$, which is a subset of diagnosis $\{pfo, bop\}$ within the other candidate item associated with observation $\mathbf{p}$. Then, at the occurrence of the first observation $\mathbf{b}$ (line 35), the candidate set $\mathbb{D}' = \{\{pfo\}\}$ is extracted from state d_2 and appended to $\mathbf{\Delta}$ (which is initially empty). At the end of the iteration (line 37), $\mathcal{G}$ and $\mathcal{U}$ are initialized with $\mathbb{I}' = \{(0, \{pfo\}), (3, \{pfo, bop\})\}$, as displayed in the first column of the second iteration in Table 2, while the current state d of $\mathcal{M}$ is set to d_2.

Iteration 2

At the second iteration, the two initial items included in $\mathcal{U}$ are the roots of the two graphs in position 2 in Figure 4, namely $(0, \{pfo\})$ and $(3, \{pfo, bop\})$. The processing of $(0, \{pfo\})$ leads to the generation of a candidate item $(1, \{pfo\}, \mathbf{p})$ in $\mathcal{C}$, and a new abduction item $(2, \{pfo\})$. The subsequent processing of $(2, \{pfo\})$ generates two candidate items, namely $(0, \{pfo\}, \mathbf{b})$ and $(3, \{pfo, bop\}, \mathbf{b})$. On its part, item $(3, \{pfo, bop\})$ generates the candidate item $(4, \{pfo, bop\}, \mathbf{p})$ and a new abduction item $(5, \{pfo, bop\})$, whose processing leads to the creation of the candidate item $(0, \{pfo, bop\}, \mathbf{b})$ and a new abduction item $(3, \{pfo, bfc, bop\})$. On the one hand, since $\mathcal{C}$ includes candidate item $(0, \{pfo\}, \mathbf{b})$, where $\{pfo\} \subset \{pfo, bop\}$, condition in line 18 is not fulfilled, thus $(0, \{pfo, bop\}, \mathbf{b})$ is ignored. On the other, since $\mathcal{G}$ includes abduction item $(3, \{pfo, bop\})$, which is the root of the second graph, where

Table 2 Data structures of Minimalist Diagnosis Engine with $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}, \mathbf{b}]$ of $\mathcal{X}_w$.

i	$\mathbb{I}$	$\mathcal{G}$	$\mathcal{C}$	o'	$\mathbb{D}'$
1	$(0, \emptyset)$	$(0, \emptyset)$ $(2, \{pfo\})$	$(1, \emptyset, \mathbf{p})$ $(0, \{pfo\}, \mathbf{b})$ $(3, \{pfo, bop\}, \mathbf{b})\}$	**b**	$\{\{pfo\}\}$
2	$(0, \{pfo\})$ $(3, \{pfo, bop\})$	$(0, \{pfo\})$ $(2, \{pfo\})$ $(3, \{pfo, bop\})$ $(5, \{pfo, bop\})$ $(3, \{pfo, bfc, bop\})$	$(1, \{pfo\}, \mathbf{p})$ $(0, \{pfo\}, \mathbf{b})$ $(3, \{pfo, bop\}, \mathbf{b})$ $(4, \{pfo, bop\}, \mathbf{p})$	**p**	$\{\{pfo\}\}$
3	$(1, \{pfo\})$ $(4, \{pfo, bop\})$	$(1, \{pfo\})$ $(4, \{pfo, bop\})$ $(7, \{pfo, bfo\})$ $(1, \{pfo, pfc, bfo\})$	$(6, \{pfo\}, \mathbf{b})$ $(2, \{pfo, bfo\}, \mathbf{p})$ $(7, \{pfo, bop, bcl\}, \mathbf{b})$	**p**	$\{\{pfo, bfo\}\}$
4	$(2, \{pfo, bfo\})$	$\{(2, \{pfo, bfo\})\}$	$(0, \{pfo, bfo\}, \mathbf{b})$ $(3, \{pfo, bfo, bop\}, \mathbf{b})$	**b**	$\{\{pfo, bfo\}\}$
5	$(0, \{pfo, bfo\})$ $(3, \{pfo, bfo, bop\})$	$(0, \{pfo, bfo\})$ $(3, \{pfo, bfo, bop\})$ $(2, \{pfo, bfo\})$ $(5, \{pfo, bfo, bop\})$ $(3, \{pfo, bfo, bfc, bop\})$	$(1, \{pfo, bfo\}, \mathbf{p})$ $(0, \{pfo, bfo\}, \mathbf{b})$ $(3, \{pfo, bfo, bop\}, \mathbf{b})$ $(4, \{pfo, bfo, bop\}, \mathbf{p})$	**b**	$\{\{pfo, bfo\}\}$

$\{pfo, bop) \subset \{pfo, bfc, bop\}$, abduction item $(3, \{pfo, bfc, bop\})$ is marked in $\mathcal{G}$ (line 13), so that it will be discarded without being processed when considered subsequently. Since now $\mathcal{U}$ is empty (line 22), the nested loop terminates. The candidate abduction items relevant to observation $\mathbf{p}$ lead to the creation of the new state d_3 in $\mathcal{M}$, along with its entering transition. Those relevant to $\mathbf{b}$, instead, lead to the set $\mathbb{I}_j = \{(0, \{pfo\}), (3, \{pfo, bop\})\}$, which identifies state d_2. Hence, the target state of the transition exiting d_2 and marked by $\mathbf{b}$ is already in $\mathcal{M}$, so that a transition $\langle d_2, \mathbf{b}, d_2\rangle$ can be created immediately in line 32. Then, at the occurrence of the second observation $\mathbf{p}$ (line 35), the candidate set $\mathbb{D}' = \{\{pfo\}\}$ is extracted from state d_3 and appended to $\mathbf{\Delta}$. At the end of the iteration (line 37), $\mathcal{G}$ and $\mathcal{U}$ are initialized with $\mathbb{I}' = \{(1, \{pfo\}), (4, \{pfo, bop\})\}$, as displayed in the first column of the third iteration in Table 2, while the current state d of $\mathcal{M}$ is set to d_3.

Iteration 3

At the third iteration, the graph on the left (Figure 4, position 3) generates two candidate items in $\mathcal{C}$: $(6, \{pfo\}, \mathbf{b})$ and $(2, \{pfo, bfo\}, \mathbf{p})$. Note that the abduction item $(1, \{pfo, pfc, bfo\})$ is pruned on the grounds that the relevant diagnosis is a superset of the diagnosis in the root (sharing the same state 1), namely $\{pfo, pfc, bfo\} \supset \{pfo\}$. The second graph (on the right) generates a candidate item $(6, \{pfo, bop\}, \mathbf{b})$, which, however, contrasts with $(6, \{pfo\}, \mathbf{b})$, which is already in $\mathcal{C}$ (cf. the first graph on the left); thus, this candidate item is not inserted into $\mathcal{C}$, as it was for the candidate item $(0, \{pfo, bop\}, \mathbf{b})$ in the second iteration. The second candidate item generated in the same graph, instead, namely $(7, \{pfo, bop, bcl\}, \mathbf{b})$, is inserted into $\mathcal{C}$. Eventually, $\mathcal{C}$ includes three candidate items (cf. third row in Table 2). When $\mathcal{U}$ becomes empty (line 22), the candidate abduction items relevant to observation $\mathbf{b}$ lead to the creation of the new state d_4 in $\mathcal{M}$, while those relevant to $\mathbf{p}$ lead to the creation of state d_5, along with corresponding entering transitions. Then, at the occurrence of the third observation $\mathbf{p}$ (line 35), the candidate set $\mathbb{D}' = \{\{pfo, bfo\}\}$ is extracted from state

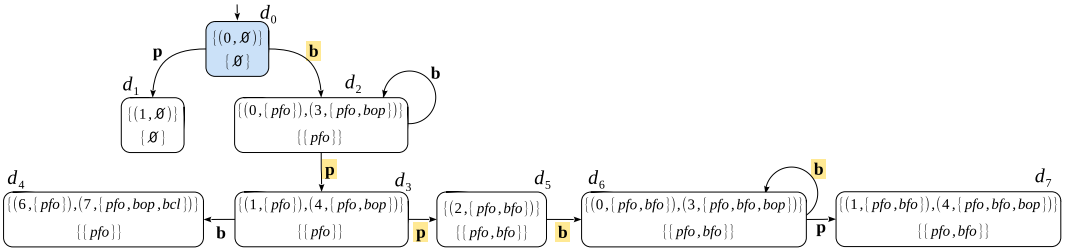


Figure 5 Lazy generation of minimalist diagnoser $Mind(\mathcal{X}_w)$ based on $\mathcal{O} = [\mathbf{b}, \mathbf{p}, \mathbf{p}, \mathbf{b}, \mathbf{b}]$ of $\mathcal{X}_w$.

d_5 and appended to $\mathbf{\Delta}$. At the end of the iteration (line 37), $\mathcal{G}$ and $\mathcal{U}$ are initialized with $\mathbb{I}' = \{(2, \{pfo, bfo\})\}$, as displayed in the first column of the fourth iteration in Table 2, while the current state d of $\mathcal{M}$ is set to d_5.

Iteration 4

At the fourth iteration, the only initial item in $\mathcal{U}$ is $(2, \{pfo, bfo\})$, which is also the root of the graph in Figure 4 (position 4). Now, since both transitions exiting state 2 in $Space(\mathcal{X}_w)$ are observable, two candidate items are generated and inserted into $\mathcal{C}$: $(0, \{pfo, bfo\}, \mathbf{b})$ and $(3, \{pfo, bfo, bop\}, \mathbf{b})$. Since $\mathcal{U}$ is now empty, the partition $\mathbf{C}$ in line 23 is in fact a singleton, which includes $\mathcal{C}$ (both candidate items in $\mathcal{C}$ are associated with observation $\mathbf{b}$). Thus, a new state d_6 is generated in $\mathcal{M}$, along with transition $\langle d_5, \mathbf{b}, d_6 \rangle$. At the occurrence of the fourth observation $\mathbf{b}$ (line 35), the candidate set $\mathbb{D}' = \{\{pfo, bfo\}\}$ is extracted from state d_6 and appended to $\mathbf{\Delta}$. At the end of the iteration (line 37), $\mathcal{G}$ and $\mathcal{U}$ are initialized with $\mathbb{I}' = \{(0, \{pfo, bfo\}), (3, \{pfo, bfo, bop\})\}$, as displayed in the first column of the fifth iteration in Table 2, while the current state d of $\mathcal{M}$ is set to d_6.

Iteration 5

In the fifth iteration, the two graphs rooted in the abduction items $(0, \{pfo, bfo\})$ and $(3, \{pfo, bfo, bop\})$, as outlined in the last row in Figure 4 and Table 2, involve the generation of abduction items $(2, \{pfo, bfo\})$, $(5, \{bfo, pfo, bop\})$, and $(3, \{pfo, bfo, bfc, bop\})$, the last of which is marked in $\mathcal{G}$ because the relevant diagnosis is a superset of the diagnosis inherent to the root, namely $\{pfo, bfo, bop\}$. Furthermore, of the five candidate items generated, $(0, \{pfo, bfo, bop\}, \mathbf{b})$ is not inserted into $\mathcal{C}$ because the latter includes candidate item $(0, \{pfo, bfo\}, \mathbf{b})$, where $\{pfo, bfo\} \subset \{pfo, bfo, bop\}$. Then, partition $\mathbf{C}$ in line 23 leads to the generation of a new state d_7 in $\mathcal{M}$, along with a new transition $\langle d_6, \mathbf{p}, d_7 \rangle$. Since, however, the part associated with observation $\mathbf{b}$ leads to $\mathbb{I}'_j = \{(0, \{pfo, bfo\}), (3, \{pfo, bfo, bop\})$, which equals the field $\mathbb{I}$ of the same state d_6, a new auto-transition $\langle d_6, b, d_6 \rangle$ is generated without the need to materialize the (existing) target state d_6. At the occurrence of the fifth observation $\mathbf{b}$ (line 35), the candidate set $\mathbb{D}' = \{\{pfo, bfo\}\}$ is extracted from state d_6 and appended to $\mathbf{\Delta}$. At the end of the iteration (line 37), the current state d of $\mathcal{M}$ is still d_6.

Since $\mathcal{O}$ includes five observations, the sequence of diagnosis sets generated by Minimalist Diagnosis Engine is $\mathbf{\Delta} = [\{\{pfo\}\}, \{\{pfo\}\}, \{\{pfo, bfo\}\}, \{\{pfo, bfo\}\}, \{\{pfo, bfo\}\}]$. Note how, at the fourth observation, $\mathbf{\Delta}$ equals the temporal candidate set anticipated in Example 13. Remarkably, as claimed in Proposition 21, this is no coincidence.

4.2 Correctness of the algorithm

▶ **Proposition 21.** *Algorithm* Minimalist Diagnosis Engine *is correct.*

Proof. According to Corollary 19, it suffices to show that the part of $Mind(\mathcal{X}) = (\mathbf{O}, D, \tau, d_0)$ materialized in $\mathcal{M}$ is correct, which can be proven by induction on the transition function τ.

(*Basis*) *The initial state d_0 generated in line 2 is correct.* In fact, $\Im_0 = (x_0, \emptyset)$, $d_0 = (\{\Im_0\}, \{\emptyset\}) = (\{(x_0, \emptyset)\}, \{\emptyset\})$, which equals the initial state of $Mind(\mathcal{X})$ in Definition 16.

(*Induction*) *If $d = (\mathbb{I}, \mathbb{D})$ is a state materialized in $\mathcal{M}$, then all transitions $\langle d, o'_j, d_j \rangle$ generated in line 32 are correct.* Based on the abduction items in $\mathbb{D}$, which have been pushed onto $\mathcal{U}$ in line 37, the nested loop in line 5–22 is bound to generate in $\mathcal{C}$ all the candidate items for every possible next observation $o'_1, \ldots, o'_k$. To this end, an unmarked abduction item (that is, an abduction item neither pruned not yet processed) at a time is popped from $\mathcal{U}$, and its successive abduction items $\Im' = (x', \delta')$, which are relevant to its frontier, are computed (line 9). If the involved component transition from x to x' is unobservable ($o = \varepsilon$), then, provided that $\Im'$ has not been generated already (line 11), some abduction items are possibly marked in lines 12–16 in order to prune the trajectories that will not contribute to the instances of $\mathbb{D}_j$ in the newly-materialized states d_j of $\mathcal{M}$. This pruning is dictated by efficiency reasons only: if not performed, the processing in lines 18–20 still allows for the eventual generation of the correct candidate set $\mathbb{D}_j$, but at the expense of useless computation of additional abduction items in $\mathbb{I}_j$. In fact, if condition in line 12 is true, then any abduction item derived from (x', δ') cannot have a diagnosis that is a subset of an abduction item derived from $(x', \bar{\delta})$, as all the additional faults involved in an extension of δ' are also involved in an extension of $\bar{\delta}$, precisely because the suffix of a trajectory starting in x' depends only on x', not on the diagnosis associated in the abduction item. If, instead, condition in line 12 is false, then we need to mark every abduction item $(x', \bar{\delta})$ where $\bar{\delta} \supset \delta'$, for the same reason of keeping only abduction items involving the same state of $\mathcal{X}$ that have the minimal diagnosis. In contrast with the processing in lines 14–16, however, we do not impose in this case the condition $x' \neq x$. In fact, in conformity with Definition 16, once a candidate item (x', δ', o) is inserted into $\mathcal{C}$ (line 19), no other successive item needs to be generated (starting from its frontier). Hence, the removal in line 20 is necessary even if $x' = x$. When $\mathcal{U}$ becomes empty in line 22 (end of the nested loop), it means that there is no other abduction or candidate item that can be generated (we have reached the completeness). In line 23, the partition $\mathbf{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ allows us to associate with each possible next observation o'_j the relevant candidate items (x, δ, o'_j). In other words, each part $\mathcal{C}_j \in \mathbf{C}$ contains the information for generating the field $\mathbb{I}_j$ of state d_j in transition $\langle d, o'_j, d_j \rangle$, which is performed in lines 24–32. Specifically, $\mathbb{I}_j$ is distilled from $\mathcal{C}_j$ and, if there is no state d_j in $\mathcal{M}$ identified by $\mathbb{I}_j$, then it is generated by first computing the set $\mathbb{D}_j$ based on Definition 16, that is, as the set of minimal diagnoses in $\mathbb{I}_j$ (lines 26–31). Eventually, a transition $\langle d, o'_j, d_j \rangle$ is generated in $\mathcal{M}$. Since, based on the argumentation expressed above, the set $\mathbb{I}_j$ of abduction items computed by the algorithm equals the first field of the target state d'_j of the transition exiting d and marked by observation o'_j, and the set $\mathbb{D}_j$ equals the second field of d'_j, the transition function of d materialized in $\mathcal{M}$ is correct. ◀

5 Related work

The work presented in this paper stems from the active system approach [13, 16], which represents DESs by means of complete explicit (i.e. operational) strong component models and performs abduction-based diagnosis. More specifically, in this paper a DES is a network

of communicating finite automata. Other explicit models for DESs are Petri nets [1, 3] and labeled time Petri nets [19]. Strong models can be adopted to represent dynamical systems also when consistency-based reasoning is performed. For instance, the diagnostic approach in [7], which is applied to synchronous sequential circuits, is consistency-based. Both the normal and abnormal behavior (fault modes) of system components are described by means of the Finite Trace Next Logic, and minimum-cardinality diagnoses are computed. Weak models and consistency-based diagnosis are applied to DESs in [18]. There are contributions about diagnostic reasoning that ignore any explicit DES models, instead, they consider some specifications. The specification of a dynamical system (i.e. the properties the system has to exhibit over time) can be given as a formula in a temporal logic [6], such as Linear Temporal Logic (LTL) [22]. An LTL specification is the implicit representation of an automaton, where LTL operators describe the state transitions. Typical diagnostic tasks are aimed at finding out whether a given behavioral evolution, called a *trace*, satisfies the specification formula and/or uncovering the causes for a trace violates the specification formula, where such causes can be searched for either in the trace [2] or in the specification [20, 21], that is, the specification may be wrong.

To the best of our knowledge, the first work that adopted subset-minimal diagnoses for DESs is [27], where, however, their appropriateness is not investigated: minimal diagnoses are reckoned good as they are "more probable" and they reduce the cognitive load of a human operator. The method in [27], differently from the proposal presented here, is based on a total offline knowledge-compilation, aimed at generating a so-called *minimal diagnoser*. In [27], computing minimal candidates only does not translate into any pruning of the search space; on the contrary, the construction of the minimal diagnoser requires the previous generation the whole DES space, which is unfeasible for real systems since the number of states is exponential in the number of components. The process in this paper, instead, performs online a (partial dynamical) knowledge-compilation, if and when needed (laziness), and every observation-driven behavior reconstruction assumes the unavailability of the DES space, which is only instrumental to the formalization, but never materialized.

6 Conclusion

From the computational viewpoint, diagnosis of real DESs is a complex task because of the huge size of the DES space in which the search for the trajectories complying with a temporal observation is carried out. This is why the total compilation of a DES into a diagnoser is impractical, even when performed offline. Also the online reconstruction of the DES trajectories based on the given temporal observation is no panacea. From the user (or post-processing) side, analyzing all the candidates is possibly time consuming. Hence, in this paper, instead of generating all the diagnoses, each being a set of faults, we generate (subset) minimal diagnoses only. First, we have discussed the meaningfulness of minimal diagnoses for task of DES diagnosis (and, more specifically, for diagnosis during monitoring). Then, we have proposed a method that, at each newly perceived observation, updates the trajectories built up to the reception of the previous observation, while pruning the trajectories that either are not consistent with the new observation or are bound to bring to non-minimal candidates, thus reducing the size of the DES space to be explored. In addition, every time a new observation is perceived, the method updates (if needed) an (initially empty) compiled knowledge structure, called a *minimalist diagnoser*, so as to speed up the next abductive reasoning steps.

The method presented in this paper comes with expectations of gains in spatial and temporal performances with respect to the computation of all diagnoses; such expectations are being confirmed by an ongoing experimental activity.

A future version of the method could generate candidates both incrementally and in a sorted way based on a preference criterion, such as cardinality or likelihood, that is significant in the considered domain so as every candidate that is preferred to another is outputted before it. Further research could adapt the proposed algorithm to notions of DES diagnosis different from that adopted in this paper. For instance, a recent work [11] defines a DES diagnosis as the *sequence* of faults associated with a trajectory, namely a *temporal fault*. The adaptation would require defining the concept of *minimal temporal fault*, representing concisely the minimal temporal fault candidates, pruning the portions of the search space that lead to non-minimal temporal faults, and progressively updating the relevant sequence-oriented minimalist diagnoser.

In the literature the notion of a fault in a DES has been generalized to the violation of a predefined behavioral property, as in [8], or to a pattern of transitions, called *supervision pattern* [10]. A fault can also be defined as an event that arises when a sub-trajectory of the DES matches a given pattern, like in [14, 17]. Minimal candidates could in principle be computed also for generalized faults; their significance, however, needs some investigation.

References

1 F. Basile. Overview of fault diagnosis methods based on Petri net models. In *Proceedings of the 2014 European Control Conference, ECC 2014*, pages 2636–2642, 2014. doi:10.1109/ECC.2014.6862631.

2 I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining counterexamples using causality. In A. Bouajjani and O. Maler, editors, *21st International Conference on Computer-Aided Verification (CAV 2009)*, pages 94–108. Springer-Verlag, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-02658-4_11.

3 X. Cong, M.P. Fanti, A.M. Mangini, and Z. Li. Decentralized diagnosis by Petri nets and integer linear programming. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(10):1689–1700, 2018. doi:10.1109/TSMC.2017.2726108.

4 J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses. In G. Gottlob and W. Nejdl, editors, *Expert Systems in Engineering, Principles and Applications*, pages 1–15. Springer, 1990. doi:10.1007/3-540-53104-1_27.

5 J. de Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

6 E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990. doi:10.1016/B978-0-444-88074-1.50021-4.

7 Alexander Feldman, Ingo Pill, Franza Wotawa, Ion Matei, and Johan de Kleer. Efficient model-based diagnosis of sequential circuits. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:2814–2821, 2020. doi:10.1609/aaai.v34i03.5670.

8 G. Göessler, T. Mari, Y. Pencolé, and L. Travé-Massuyès. Towards causal explanations of property violations in discrete event systems. In *30th International Workshop on Principles of Diagnosis (DX-19)*, Klagenfurt, Austria, 2019. URL: https://api.semanticscholar.org/CorpusID:226220644.

9 W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, CA, 1992.

10 T. Jéron, H. Marchand, S. Pinchinat, and M.O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems (WODES 2006)*, pages 262–268, Ann Arbor, MI, 2006. IEEE Computer Society. URL: https://api.semanticscholar.org/CorpusID:526355.

11 G. Lamperti, S. Trerotola, M. Zanella, and X. Zhao. Sequence-oriented diagnosis of discrete-event systems. *Journal of Artificial Intelligence Research*, 78:69–141, 2023. doi:10.1613/jair.1.14630.

12 G. Lamperti and M. Zanella. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, 137(1–2):91–163, 2002. doi:10.1016/S0004-3702(02)00123-6.

13 G. Lamperti and M. Zanella. *Diagnosis of Active Systems: Principles and Techniques*, volume 741 of *Springer International Series in Engineering and Computer Science*. Springer, Dordrecht, Netherlands, 2003. doi:10.1007/978-94-017-0257-7.

14 G. Lamperti and M. Zanella. Context-sensitive diagnosis of discrete-event systems. In T. Walsh, editor, *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, volume 2, pages 969–975, Barcelona, Spain, 2011. AAAI Press. doi:10.5591/978-1-57735-516-8/IJCAI11-167.

15 G. Lamperti and M. Zanella. Monitoring of active systems with stratified uncertain observations. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 41(2):356–369, 2011. doi:10.1109/TSMCA.2010.2069096.

16 G. Lamperti, M. Zanella, and X. Zhao. *Introduction to Diagnosis of Active Systems*. Springer, Cham, 2018. doi:10.1007/978-3-319-92733-6.

17 G. Lamperti and X. Zhao. Diagnosis of active systems by semantic patterns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(8):1028–1043, 2014. doi:10.1109/TSMC.2013.2296277.

18 Y. Pencolé, G. Steinbauer, C. Mühlbacher, and L. Travé-Massuyès. Diagnosing discrete event systems using nominal models only. In M. Zanella, I. Pill, and A. Cimatti, editors, *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4, pages 169–183. Kalpa Publications in Computing, 2018. doi:10.29007/1d2x.

19 Y. Pencolé, A. Subias, and C. Coquand. A model checking method to solve the event pattern diagnosis problem in safe labeled time Petri nets. In *32nd International Workshop on Principles of Diagnosis (DX-21)*, Hamburg, Germany, 2021. URL: https://laas.hal.science/hal-03348338v1.

20 Ingo Pill and Thomas Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 1053–1059. AAAI Press, 2013. URL: http://dl.acm.org/citation.cfm?id=2540128.2540280.

21 Ingo Pill and Franz Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139(C):124–141, 2018. doi:10.1016/J.JSS.2018.02.002.

22 Amir Pnueli. The temporal logic of programs. In *8th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi:10.1109/SFCS.1977.32.

23 D. Poole. Normality and faults in logic-based diagnosis. In *Eleventh International Joint Conference on Artificial Intelligence (IJCAI 1989)*, pages 1304–1310, Detroit, MI, 1989. URL: http://ijcai.org/Proceedings/89-2/Papers/073.pdf.

24 R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

25 Patrick Rodler. *How Should I Compute My Candidates? A Taxonomy and Classification of Diagnosis Computation Algorithms*, pages 1986–1993. IOS Press, 2023. doi:10.3233/faia230490.

26 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995. doi:10.1109/9.412626.

27 X. Zhao, D. Ouyang, G. Lamperti, and X. Tong. Minimal diagnosis and diagnosability of discrete-event systems modeled by automata. *Complexity*, 2020:1–17, 2020. doi:10.1155/2020/4306261.

MSO Sets and MTES for Dummies

Maxence Glotin
LAAS-CNRS, Université de Toulouse, INSA, France

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, France

Elodie Chanthery
LAAS-CNRS, Université de Toulouse, INSA, France

Abstract

Structural analysis-based diagnosis allows for the extraction of a wealth of information and properties by studying a structural model that represents a physical system. This diagnosis approach is centered on structurally overdetermined sets, which enable the generation of residuals for fault detection and isolation. As the 'for Dummies' editorial collection, this article aims at taking on complex concepts and making them easy to understand. It aims to clarify and compare key concepts in structural analysis, focusing on Minimally Structurally Overdetermined (MSO) sets and Minimal Test Equation Supports (MTES). Additionally, we explain and illustrate the Dulmage-Mendelsohn decomposition, which helps identify structurally overdetermined parts of the system and plays a important role in the structural analysis process. Through detailed exploration and practical examples, we demonstrate the roles, applications, and interrelations of these sets, highlighting their respective strengths and limitations. The paper provides an overview of the algorithms used to identify and use these sets, including a theoretical and practical comparison of their computational efficiency and diagnostic capabilities.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Structural analysis, MTES, MSO sets

Digital Object Identifier 10.4230/OASIcs.DX.2024.13

Funding This work is partially supported by the Artificial and Natural Intelligence Toulouse Institute (ANITI), French "Investing for the Future - PIA3" program under the Grant agreement ANR-19-PI3A-0004.

1 Introduction

In the field of model-based fault diagnosis, the ability to detect and isolate faults within complex systems is crucial for ensuring reliability and safety. By studying the model, it is possible to identify faults, which are indicative of abnormal or faulty behavior within the system.

Structural analysis consists in abstracting a model by keeping only the links between equations and variables. The main advantages of structural analysis are that the approach can be applied to large scale systems, linear or non linear, even under uncertainty. In the field, the concept of Minimally Structurally Overdetermined (MSO) sets [6] has been widely adopted due to its effectiveness in isolating faults by leveraging the redundancy in system models. However, while MSO sets have proven valuable, they come with limitations, particularly in terms of computational complexity when applied to large-scale systems.

To address these issues, more refined approaches have been introduced, such as Minimal Test Equation Supports (MTES), presented in [5], or Fault Driven Minimal Structurally Overdetermined (FMSO) sets introduced by [7]. These methods aim to enhance diagnosis precision while aiming at reducing computational demands, yet their practical application and comparative advantages remain areas of ongoing exploration.


© Maxence Glotin, Louise Travé-Massuyès, and Elodie Chanthery;
licensed under Creative Commons License CC-BY 4.0



As the 'for Dummies' editorial collection, this article aims at taking on complex concepts and making them easy to understand. The objective is to clarify and compare various key concepts used in structural analysis model-based fault diagnosis, specifically focusing on MSO sets, MTES, or FMSO sets. By providing a detailed exploration of these concepts, this paper seeks to clarify their roles, applications, and interrelations. Through carefully chosen examples, we illustrate how each of these concepts can be applied to different scenarios, highlighting their respective strengths and limitations. This comparative analysis is intended to enhance the understanding of these diagnosis tools.

This paper is structured as follows. Section 1 gives some theoretical foundations on structural analysis. Then in Section 3 MSO sets are defined and explained. The algorithm for their identification is also presented. Next, Section 4 discuss the motivations behind the introduction of MTES and describe the algorithm for their computation. A comparison between the performance of MSO and MTES is then conducted in Section 5. To illustrate the practical application of these concepts, a realistic example is provided in Section 6, including structural modeling, algorithmic complexity analysis, and an evaluation of diagnostic capability. Section 7 concludes the paper.

2 Background

One of the key concepts to understand the notion of MSO sets or even MTES is the Dulmage Mendelsohn decomposition of a bipartite graph [4].

Let the system description consist of a set of n_e equations involving a set of variables partitioned into a set Z of n_Z known (or measured) variables and a set X of n_X unknown (or unmeasured) variables. We refer to the vector of known variables as z and the vector of unknown variables as x. The system may be impacted by the presence of n_f faults that appear as parameters in the equations. The set of faults is denoted by F and we refer to the vector of faults as $\boldsymbol{f}$.

▶ **Definition 1** (System). *A system, denoted $\Sigma(z, x, \boldsymbol{f})$ or Σ for short, is any set of equations relating z, x and $\boldsymbol{f}$. The equations $e_i(z, x) \subseteq \Sigma(z, x, \boldsymbol{f})$, $i = 1, \ldots, n_e$, are assumed to be differential or algebraic in z and x.*

Consider the illustrative example given by system (1), from [8], named Σ_1 that will be taken as a running example in this section. It is composed of six equations e_1 to e_6 relating the known variables $Z = \{z_1, z_2\}$, the unknown variables $X = \{x_1, x_2, x_3, x_4, x_5\}$ and the set of system faults $F = \{f_1, f_2\}$. Besides a, b, c are constant parameters.

$$
\begin{aligned}
e_1 &: \quad \dot{x}_3 = e^{x_3} - a \\
e_2 &: \quad x_3^2 = -b\dot{x}_4 + f_1 \\
e_3 &: \quad z_1 = x_4 \\
e_4 &: \quad z_2 = x_1 + b^2 + x_4 \\
e_5 &: \quad \dot{x}_1 = e^{x_2} + x_5 \\
e_6 &: \quad \dot{x}_3 = x_4 + c + f_2
\end{aligned}
\tag{1}
$$

We can represent a system $\Sigma(z, x, \boldsymbol{f})$ by a biadjacency matrix, in which each row stands for an equation and each column an unknown variable. An X in position (i, j) means that equation i contains the unknown variable j. For example, Σ_1 can be represented by the biadjacency matrix of Table 1.

Table 1 Biadjacency matrix of Σ_1.

	x_1	x_2	x_3	x_4	x_5
e_1			×		
e_2			×	×	
e_3				×	
e_4	×			×	
e_5	×	×			×
e_6			×	×	

Equivalently, a system $\Sigma(z, x, f)$ can be represented by a bipartite graph $G(E, X)$. E and X are the sets of nodes, an edge between node E_i and node X_j means that equation i contains the unknown variable j. The bipartite graph associated with the illustrative example Σ_1 is shown in Figure 1 [8].

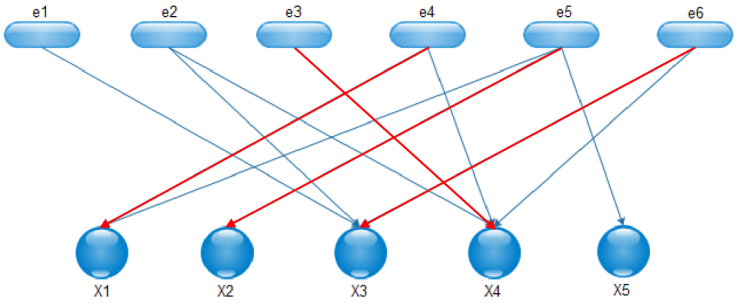


Figure 1 The bipartite graph of Σ_1 [8].

▶ **Definition 2** (Maximal Matching). *A maximal matching is a set of edges such that these edges do not share any common nodes, and no more edges can be added to this set without losing this property.*

In Table 1, as well as in Figure 1, a maximal matching is marked in red. It is the largest set of edges such that there are no common nodes (each equation and each variable appears at most once).

▶ **Definition 3** (Free equation). *For a given maximal matching, an equation is free if it is not part of this maximal matching.*

In the matching represented in Table 1, e_1 and e_2 are free.

▶ **Definition 4** (Free variable). *For a given maximal matching, a variable is free if it is not part of this maximal matching.*

In the matching represented in Table 1, x_5 is free.

▶ **Definition 5** (Alternating path). *Let M_{max} be a maximal matching. P, a path in the graph G, is an alternating path if it successively passes through edges that belong to M_{max} and edges that do not belong to M_{max}.*

The Dulmage-Mendelsohn decomposition allows separating a model Σ into 3 parts: an overdetermined part, denoted Σ^+, an exactly determined part, denoted Σ^0, and an underdetermined part, denoted Σ^-, as shown in Figure 2 .

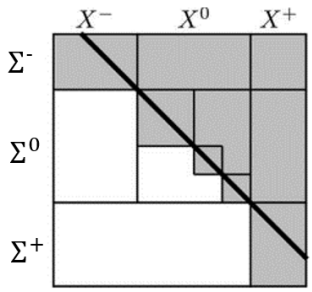


Figure 2 Dulmage-Mendelsohn decomposition of a model Σ.

The overdetermined part Σ^+ of Σ is the set of equations $e \in E$ such that, for any maximal matching, there exists an alternating path between at least one free equation and e [6]. Σ^+ can also be defined as the set of equations $e \in E$ such that there is at least one maximal matching in which e is free.

The part Σ^0 contains equations that are not in Σ^+ but are such that the set $\Sigma^0 \cup \Sigma^+$ is the maximal set of equations for which there exists a complete matching of the unknown variables in $\Sigma^0 \cup \Sigma^+$ into $\Sigma^0 \cup \Sigma^+$ [6]. We can also say that Σ^0 consists of the equations e that are matched in every maximal matching and do not contain any free variables.

The rest of the equations belongs to Σ^-. Σ^- only contains equations if the maximal matchings do not cover all the variables (if there exists at least one free variable in a maximal matching). Σ^- contains equations associated with the free variables.

Running example: Σ_1^+ contains all the equations that are free in at least one maximal matching. In the matching represented in Table 1, e_1 and e_2 are free so these two equations are in Σ_1^+. But $(e_1x_3, e_2x_4, e_4x_1, e_5x_2)$ is also a maximal matching, and in this matching, e_3 and e_6 are free. So e_3 and e_6 are also in Σ_1^+. So, we have $\Sigma_1^+ = \{e_1, e_2, e_3, e_6\}$.

$\Sigma_1^0 = \{e_4\}$ because the set $\{e_1, e_2, e_3, e_4, e_6\}$ is the biggest set such that there exists a complete matching of the unknown variables in this set into this set. If we add e_5 in that set, it would not be a complete matching because either x_2 or x_5 would not be part of the matching. Another way to see this is that e_4 is necessary to every maximal matching, as it is the only equation that contains x_1 (e_5 also contains x_1, but e_5 is used to cover x_2 or x_5). Thus, it is impossible to obtain a maximal matching without e_4, and e_4 does not contain any free variable, so e_4 is part of Σ_1^0.

The remaining equation, e_5, is in Σ_1^-. That is because e_5 is the only equation that contains both x_2 and x_5, so for every maximal matching, one of these two variables is free.

Note that, as shown above with Σ_1, if a set of equations Σ has more equations than unknown variables, it does not necessarily mean that Σ^0 and Σ^- are empty.

▶ **Definition 6** (PSO set). *A set Σ of equations is a PSO set (proper structurally overdetermined set) if $\Sigma = \Sigma^+$. In other words, Σ is a PSO set if $\Sigma^0 = \Sigma^- = \emptyset$.*

System represented Σ_1 as a whole is not a PSO set because Σ^0 and Σ^- are not empty. But the subset of equations $\{e_1, e_2, e_3\}$ is a PSO set: the three equations are free in at least one maximal matching, so $\Sigma = \Sigma^+$ (see Table 2).

Table 2 The subset of equations $\{e_1, e_2, e_3\}$.

	x_3	x_4
e_1	×	
e_2	×	×
e_3		×

▶ **Definition 7** (Structural redundancy). *For a given set of equations Σ, the structural redundancy $\varphi(\Sigma)$ is defined by $\varphi(\Sigma) = |\Sigma^+| - |X^+|$, where $|\Sigma^+|$ is the number of equations in Σ^+, and $|X^+|$ is the number of unknown variables involved in Σ^+.*

PSO sets have structural redundancy, which is key to diagnosis since it allows residual generation. We recall hereafter some important definitions [8].

▶ **Definition 8** (ARR for $\Sigma(z, x, \mathtt{f})$). *Let $\Sigma(z, x, \boldsymbol{f})$ be a system. Then, a relation of the form $arr(z, \dot{z}, \ddot{z}, ...) = r$ is an Analytical Redundancy Relation (ARR) for $\Sigma(z, x, \boldsymbol{f})$ if for each z consistent with $\Sigma(z, x, \boldsymbol{f})$ the relation evaluation is $r = 0$, i.e., the ARR is satisfied.*

▶ **Definition 9** (Residual Generator for $\Sigma(z, x, \mathtt{f})$). *A system taking a subset of the variables z as input, and generating a scalar signal r as output, is a residual generator for the model $\Sigma(z, x, \boldsymbol{f})$ if, for all z consistent with $\Sigma(z, x, \boldsymbol{f})$, it holds that $\lim_{t\to\infty} r(t) = 0$.*

ARRs are residual generators and are used to verify the consistency of observations with the system model. The ARRs hold true if the observed behavior of the system meets the model constraints. They can be derived from the system model by eliminating unknown variables [7]. PSO sets can be used to generate ARRs as they identify the redundant parts of the system's model.

3 MSO sets

It is often more practical to focus on smaller, manageable subsets of the PSO sets that still retain the necessary redundancy for effective fault diagnosis.

▶ **Definition 10** (MSO set). *A PSO set Σ is an MSO set (minimal structurally overdetermined) if no proper subset is a PSO set. MSO sets are PSO sets such that $\varphi(\Sigma) = 1$.*

MSO sets are interesting because they are just overdetermined and exactly identify a set of equations from which a residual generator can be obtained. [6].

In our running example, the subset $\{e_1, e_2, e_3\}$ is an MSO set because it is a PSO set, and there is no subset which is also a PSO set. Furthermore, $\varphi(\Sigma) = 3 - 2 = 1$.

A deep first search algorithm[1] to find MSO sets is presented in [6]. The general idea is to start with a PSO set Σ, to remove an equation e, and to compute the overdetermined part of the remaining set $(\Sigma \setminus \{e\})^+$. When an equation e is removed from a PSO set, the structural redundancy decreases by one. So the previous process is done until the structural redundancy becomes one. If $\varphi((\Sigma \setminus \{e\})^+) = 1$, then an MSO set is found.

Running example: To illustrate that, let us apply the first steps of the algorithm to the following example Σ_2 (cf. Table 3).

[1] All the algorithms presented in this article are provided in the "Fault Diagnosis Toolbox" available here: `https://faultdiagnosistoolbox.github.io/`

Table 3 Illustrative example Σ_2.

	x_1	x_2	x_3
e_1	×		
e_2		×	×
e_3	×	×	
e_4			×
e_5	×		×

The set $\Sigma_2 = \{e_1, e_2, e_3, e_4, e_5\}$ is a PSO set of structural redundancy 2. First, e_1 is removed. By applying the Dulmage-Mendelsohn decomposition on the remaining set, we have $(\Sigma_2 \setminus \{e_1\})^+ = \{e_2, e_3, e_4, e_5\}$. $\varphi((\Sigma_2 \setminus \{e_1\})^+) = 1$, which means that $\{e_2, e_3, e_4, e_5\}$ is an MSO set. Then, e_2 is removed from Σ_2. By applying the Dulmage-Mendelsohn decomposition on the remaining set, we have $(\Sigma_2 \setminus \{e_2\})^+ = \{e_1, e_4, e_5\}$. $\varphi((\Sigma_2 \setminus \{e_2\})^+) = 1$, which means that $\{e_1, e_4, e_5\}$ is also an MSO set. And so on, until covering the entire tree.

Using this algorithm, all the MSO sets are found. However, this method is not optimal in terms of computation time because some MSO sets can be found multiple times. For example, here, when removing e_2, $(\Sigma_2 \setminus \{e_2\})^+) = (e_1, e_4, e_5)$ and when removing e_3, $(\Sigma_2 \setminus \{e_3\})^+) = (e_1, e_4, e_5)$. Therefore, the same MSO set is found twice. That is why the concept of equivalence class is introduced.

▶ **Definition 11** (Equivalence class). *Let Σ be a set of equations and (e_i, e_j), two equations in Σ. e_i and e_j are in the same equivalence class $[e] = \{e_i, e_j\}$ if $e_i \notin (\Sigma \setminus \{e_j\})^+$.*

This enables grouping equations into the same class such that, if they are removed from Σ, the same overdetermined part is obtained.

For the system Σ_2, the equivalences classes are $\{e_1\}$, $\{e_2, e_3\}$, $\{e_4\}$, $\{e_5\}$ because $e_2 \notin (\Sigma_2 \setminus \{e_3\})^+$ (the inverse is also true because it is an equivalence relation).

Thus, in the algorithm, it is no longer one single equation that is removed at each step, but equivalence classes, which prevent the same overdetermined part from being computed twice. At each step, equations that belong to the same equivalence class are lumped, one equivalence class is removed, and the overdetermined part of the remaining equations is computed. Note that for Σ_2, this optimization does not make a huge difference in terms of computation time, but it makes a big difference for huge models, with many equations and MSO sets.

There is another issue handled by the algorithm. The same MSO set can be obtained if the order of equations removal is permuted. Let us consider the following system Σ_3 (cf. Table 4).

Table 4 Illustrative example Σ_3.

	x_1	x_2	x_3
e_1		×	
e_2	×	×	×
e_3	×		
e_4		×	×
e_5	×		
e_6			×

At the beginning, e_1 is removed. The overdetermined part of the remaining equations $(\Sigma_3 \setminus \{e_1\})^+$ is $\{e_2, e_3, e_4, e_5, e_6\}$. Then, e_2 is removed, and the overdetermined part of the remaining equations $(\Sigma_3 \setminus \{e_1, e_2\})^+$ is $\{e_3, e_5\}$, which is an MSO set. But if e_2 is removed first, and then e_1, the same MSO set is obtained. To handle this problem, the algorithm uses a new input set R which contains the equivalence classes that are allowed to be removed.

At each node traversed by the algorithm, a Dulmage-Mendelsohn decomposition is performed. This is the most time-consuming part of the computation. This algorithm traverses a search tree, where each node represents a potentially overdetermined set of equations. Each level of the tree corresponds to the removal of an equation from the system, and each branch represents a subset of equations being tested. When an equation is removed from Σ (i.e., when moving down a level in the search tree), the structural redundancy $\varphi(\Sigma)$ decreases by 1 [6].

Consider the worst-case scenario, where each equation belongs to a different equivalence class (i.e., the equations are removed from Σ one by one). For a system with n_e equations and a redundancy degree of φ, the maximum number of nodes the algorithm might traverse is given by the sum of possible equation subset for each level k of the tree. A bound on the number of nodes traversed by the algorithm to find MSO sets is then given by Equation (2).

$$\sum_{k=0}^{\varphi-1} \binom{n_e}{k} \tag{2}$$

MSO sets are used to construct ARRs, and thus residual generators, but it is important to note that MSO sets do not take faults into account. If we are interested in system performance and we want to construct residual generators for diagnosis purposes, only MSO sets that contain equations possibly disrupted by faults are important.

▶ **Definition 12** (Detectable fault). *A fault f is structurally detectable in a set of equations Σ if f is in the overdetermined part Σ^+.*

▶ **Definition 13** (Fault support). *The fault support $F_{\Sigma'}$ of a set of equations $\Sigma' \subseteq \Sigma$ is defined as the set of faults that are involved in the equations of Σ' [8].*

A special class of MSO sets can be defined: FMSO sets (Fault-Driven Minimal Structurally Overdetermined) that are MSO sets with a non empty fault support [8].

▶ **Definition 14** (FMSO set). *A subset of equations $\Sigma^{'} \subseteq \Sigma(z, x, \boldsymbol{f})$ is an FMSO set of $\Sigma(z, x, \boldsymbol{f})$ if (1) $F_{\Sigma'} \neq \emptyset$ and $\varphi(\Sigma^{'}) = 1$ that means $|\Sigma^{'}| = |X_{\Sigma'}| + 1$, (2) no proper subset of $\Sigma^{'}$ is overdeterminated.*

Therefore, if there exists a FMSO set in Σ which contains the fault f, then f is detectable in Σ. Those special MSO sets are candidates as residual generators as they contain just the necessary redundancy and their fault support is not empty.

4 MTES

4.1 Motivation

MSO sets can be used to obtain residual generators, but are they all useful ? Let us take the previous system Σ_3 and add faults of the set $F = \{f_1, f_2, f_3\}$ (cf. Table 5).

Table 5 System Σ_3 with faults.

	x_1	x_2	x_3	
e_1		×		f_1
e_2	×	×	×	f_2
e_3	×			
e_4		×	×	
e_5	×			
e_6			×	f_3

For this example, there are 8 MSO sets given in Table 6.

Table 6 MSO sets and their fault support for Σ_3 with faults.

	MSO sets	Faults
MSO_1	$\{e_3, e_5\}$	$\{\emptyset\}$
MSO_2	$\{e_2, e_4, e_5, e_6\}$	$\{f_2, f_3\}$
MSO_3	$\{e_2, e_3, e_4, e_6\}$	$\{f_2, f_3\}$
MSO_4	$\{e_1, e_4, e_6\}$	$\{f_1, f_3\}$
MSO_5	$\{e_1, e_2, e_5, e_6\}$	$\{f_1, f_2, f_3\}$
MSO_6	$\{e_1, e_2, e_4, e_5\}$	$\{f_1, f_2\}$
MSO_7	$\{e_1, e_2, e_3, e_6\}$	$\{f_1, f_2, f_3\}$
MSO_8	$\{e_1, e_2, e_3, e_4\}$	$\{f_1, f_2\}$

MSO_2 and MSO_3 as well as MSO_6 and MSO_8 have the same fault support $\{f_2, f_3\}$ and $\{f_1, f_2\}$, respectively. This means that the generated residuals would be sensitive to the same faults. Note also that if the equations of MSO_2 and MSO_3 are put together, the fault support of the resulting subset of equations remains $\{f_2, f_3\}$. In addition, some MSO sets have a non minimal fault support like MSO_5 whose fault support is $\{f_1, f_2, f_3\}$ that includes the fault support of MSO_3 and MSO_4. This indicates that it is not necessary to compute all the MSO sets to obtain a parsimonious set of residual generators guaranteeing maximal diagnosability.

The above remark tells us that we need to change the perspective and focus on the possible fault supports of MSO sets rather than the MSO sets themselves. Hence the following definitions from [5] focusing on fault supports.

▶ **Definition 15** (Test Support (TS)). *Given a model Σ and a set of faults $\mathcal{F}$, a subset of faults $\zeta \subseteq \mathcal{F}$ is a TS if there exists a PSO set $\Sigma' \subseteq \Sigma$ such that $F_{\Sigma'} = \zeta$.*

▶ **Definition 16** (Minimal Test Support (MTS)). *Given a model Σ, a test support is an MTS if no proper subset is a test support.*

With the two above definition, it is now necessary to identify the subset of equations that correspond to TSs and MTSs.

▶ **Definition 17** (Test Equation Support (TES)). *A PSO set Σ is a TES if (1) $F_\Sigma \neq \emptyset$ and (2) for any PSO set Σ' such that $\Sigma' \supsetneq \Sigma$, it holds that $F_{\Sigma'} \supsetneq F_\Sigma$.*

▶ **Definition 18** (Minimal Test Equation Support (MTES)). *A TES Σ is an MTES if there is no subset of Σ that is also a TES.*

The idea of MTES is to identify the PSOs with maximal number of equations that have a given test support [5]. In other words, for a given test support ζ, it is important to identify the maximal set of equations $\Sigma^{'} = \Sigma^{'+}$ such that $F_{\Sigma'} = \zeta$, thus avoiding that two MTES have the same TS.

It is also important to pay attention to the minimality property, i.e., to identify MTES instead of simply TES. Indeed minimality avoids obtaining redundant residual generators. This can be proved by showing that, considering the set of MTES for a given system, an ARR generated from a given MTES is never redundant with respect to the set of ARRs generated from the other MTESs. This statement holds true based on the following proposition, which is proved in [3], specifically due to the second condition.

▶ **Proposition 19.** *A given ARR_i is redundant with respect to a set of ARR_s, $i \in I$, $j \notin I$, if and only if there exists $I' \subseteq I$ such that:*

1. *$\forall z(observations)$, if all ARR_{i_s}, $i \in I'$, are satisfied by z, then ARR_j is satisfied by z.*
2. *The fault support of ARR_j contains the fault support of each ARR_i, $i \in I'$:*

$$\bigcup_{i \in I'} F_{ARR_i} \subseteq F_{ARR_j}.$$

In summary, the fault supports of the MTESs are sufficient to guarantee maximal diagnosability because they correspond to all the possible MTS, i.e. to all the fault supports of possible MSOs. Every MTES has a different MTS and no MTES is redundant.

Interestingly, the set of MTES is much smaller than the set of MSO sets, which results in computation time saving.

4.2 MTES based diagnosis process

The following steps, applied on system Σ_3 illustrate the process of diagnosis based on MTES.

Step 1: Specification

The first step involves defining the isolability requirements for the faults in the system. The pair of faults that need to be isolated are specified, for example:

- I_1: isolate f_1 from f_2.
- I_2: isolate f_1 from f_3.
- I_3: isolate f_2 from f_3.

Step 2: Compute the set of MTES

MTESs are computed for the system Σ_3:

- $MTES_1 = \{e_2, e_3, e_4, e_5, e_6\}$ with corresponding MTS $\{f_2, f_3\}$ and $\varphi(MTES_1) = 2$.
- $MTES_2 = \{e_1, e_3, e_4, e_5, e_6\}$ with corresponding MTS $\{f_1, f_3\}$ and $\varphi(MTES_1) = 2$.
- $MTES_3 = \{e_1, e_2, e_3, e_4, e_5\}$ with corresponding MTS $\{f_1, f_2\}$ and $\varphi(MTES_1) = 2$.

Step 3: Find the subset of MTES that satisfies all the isolability specifications

In this step, we check if there exists a Minimal Hitting Set that satisfies the isolability conditions defined in Step 1. We examine whether each specification can be satisfied by finding a set of equations where the faults can be isolated according to the conditions.

- For I_1 (isolate f_1 from f_2), an MTES whose corresponding MTS includes f_1 and not f_2 is required. It could be $MTES_2$.

- For I_2 (isolate f_1 from f_3): an MTES whose corresponding MTS includes f_1 and not f_3 is required. It could be $MTES_3$.
- For I_3 (isolate f_2 from f_3): an MTES whose corresponding MTS includes f_2 and not f_3 is required. $MTES_3$ also works.

Step 4: Find an FMSO within each selected MTES

- $MTES_2$ includes only $FMSO_4$ that can be used as residual generator.
- $MTES_3$ includes $FMSO_6$ and $FMSO_8$, so any of those could be used as residual generator.

4.3 Algorithm

An algorithm for computing the set of MTES is presented in [5]. The concept behind this algorithm is similar to the one used to find MSO sets. In each step, an equation is removed from the model, and the overdetermined part of the remaining set is recalculated. The main difference is that instead of removing all equations successively (or equivalence classes in the improved version), only equations associated with faults are removed. By doing so, the fault support of the resulting set decreases at each iteration, aligning with our goal of finding minimal fault supports.

The algorithm begins with a TES (the largest PSO set, ensuring no larger PSO set has the same fault support), and remains within the class of TES by progressively removing the fault-carrying equations. Thus, as these equations are removed, it is ensured that the resulting set continues to satisfy the properties of a TES.

The stopping condition for this algorithm is as follows: a TES Σ is considered an MTES if all fault-carrying equations in Σ belong to the same equivalence class. Indeed, when this condition is met, if the process goes on by removing a fault-carrying equation and recalculating the overdetermined part, the resulting equation set will no longer contain any fault-carrying equations. This stopping condition is tested at each iteration, and when it is satisfied, the TES Σ is confirmed as an MTES.

Another stopping condition is given in [5]. A TES $\Sigma^{'} \subseteq \Sigma$ is an MTES if and only if $\varphi(\Sigma^{'}) = \varphi((\Sigma \setminus \Sigma_f)^{+}) + 1$, with Σ_f the set of equations in Σ affected by faults. $(\Sigma \setminus \Sigma_f)^{+}$ represents the equations that can be added without modifying the fault support, while remaining within the class of PSO sets. If we take any TES, it can be extended with the equations from $(\Sigma \setminus \Sigma_f)^{+}$. The consequence of this condition is that all MTES have the same structural redundancy (for Σ_3, the 3 MTES have structural redundancy equal to 2). Thus, either all MTES are MSO sets, i.e., their structural redundancy is 1, or none of them is exactly an MSO set because their structural redundancy is more than 1, in which case MTES obviously include several MSO sets. If all the equations of a system are affected by faults, then MTES are exactly MSO sets. The advantage of using this condition is that it avoids performing a Dulmage-Mendelsohn decomposition at each step.

If n_f is the number of fault-carrying equations, the number of nodes traversed by the search algorithm is bounded by Equation (3).

$$\sum_{k=0}^{\varphi-1} \binom{n_f}{k}. \tag{3}$$

Running example: We now apply the algorithm step by step on the illustrative example Σ_4 presented in Table 7.

Table 7 Illustrative example Σ_3.

	x_1	x_2	x_3	x_4	x_5	
e_1		×	×	×		f_1
e_2	×	×				
e_3		×				f_2
e_4			×			f_3
e_5				×	×	
e_6	×					
e_7		×				
e_8					×	
e_9			×			

First, we consider the entire set. It is a PSO set, so there is no need to compute the overdetermined part. The equivalence classes are: $\{e_1, e_5, e_8\}, \{e_2, e_6\}, \{e_3\}, \{e_4\}, \{e_7\}, \{e_9\}$. The stopping condition is not verified because all fault-carrying equations are not in the same class. Thus, $\{e_1, e_2, e_3, e_4, e_5,\ e_6, e_7, e_8, e_9\}$ is only a TES, but not a MTES. Then, an equivalence class containing a fault (f_1) is removed. The set $\{e_1, e_5, e_8\}$ is eliminated (equivalent to removing e_1 and then computing the overdetermined part). The remaining set is $\{e_2, e_3, e_4, e_6, e_7, e_9\}$. The equivalence classes are as follow: $\{e_2, e_6\}, \{e_3\}, \{e_7\}, \{e_4, e_9\}$. Since the stopping condition is still not met, we proceed by removing the equivalence class containing f_2. The resulting set is $\{e_2, e_4, e_6, e_7, e_9\}$. This time, the stopping condition is satisfied (only one fault remaining), $MTES_1$ is identified. The same process is applied to find all the MTES.

5 Summary

To sum up the previous sections, MSO sets are sets of equations in a model where the number of equations exceeds the number of unknowns by one. This overdetermination is minimal, meaning that if any equation was removed, the set would no longer be overdetermined.

MSO sets are particularly useful in fault detection as they provide the basis for producing residual generators. They retain the necessary and sufficient redundancy. However, as seen in the previous sections, relying on the generation of MSO sets to generate the residual generators that guarantee maximal diagnosability has some drawbacks. The first point is that some MSO sets may not be sensitive to faults, making those irrelevant for fault detection and diagnosis. Second, even if they can be generated systematically, finding all MSO sets can be computationally expensive for large systems. Last but not least, they do not provide any information about fault isolation.

FMSO sets are MSO sets but they only retain MSO sets with non-empty fault support, so they are all relevant for fault detection and isolation. They are less numerous than MSO sets, which reduces significantly computation time. However, as shown in Section 4.1, several FMSO sets may have the same fault support and some may be redundant.

By focusing on fault supports rather than specific sets of equations, TESs effectively identify exactly those fault supports that correspond to all the possible MSO sets. MTESs gather all the equations from which all the FMSO sets with the corresponding fault support can be generated. As a result, MTESs inherently address the problem of test selection. In addition, generating MTES is obviously less computationally demanding than generating MSO sets or FMSO sets. Note however that, once MTESs are obtained, one FMSO set must be extracted for each of them to achieve maximal diagnosability.

One should note that for some problem, in particular test selection for distributed systems, having all the MSO sets at hand is an advantage because subsystem interconnections express in terms of equations so equation subsets are as important as their fault supports [1, 7].

Table 8 gives a summary of the comparison of MSO sets, FMSO sets, and MTES.

Table 8 Summary comparison of MSO sets, FMSO sets, and MTES.

Aspect	MSO sets	FMSO sets	MTES
Diagnosis target	residual generation	fault driven residual generation	isolability driven residual generation
Scalability	too complex for large systems	too complex for large systems	good scalability

6 Application of the concepts on a real example: the ADCS

In this section, all the concepts discussed earlier are explored through a realistic example: the Attitude Determination and Control System (ADCS) of a Low Earth Orbit satellite [2].

6.1 Structural model of the ADCS

In this example, the dynamics of a satellite's motion are modeled using equations that describe the behavior of torque and angular velocity. The total torque acting on the satellite, T, is the sum of the torques generated by the magnetorquers, reaction wheels, and external disturbances. The rotational dynamics are influenced by the satellite's moment of inertia I and angular velocity ω.

The global system is divided into 2 main parts: the Attitude Determination System (ADS) and Attitude Control System (ACS). The ADS uses a combination of gyroscopes and vector sensors (such as sun and star sensors) to estimate the satellite's attitude state, including pitch, roll, and yaw angles, along with their corresponding rates. The resulting state vector for the satellite is $X = [\psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi}]$. These estimates are then fused together to provide feedback to the ACS, which controls actuators like reaction wheels and magnetorquers to adjust the satellite's attitude.

The structural model of the satellite's ADCS is enhanced by incorporating fault information directly into the system's equations, with faults modeled as signals. The model considers faults affecting the rate and vector sensors of the ADS (respectively *frs* and *fvs*), as well as the reaction wheels of the ACS (*frw*), including various types such as hard, soft, and intermittent faults. Each fault can affect all three axes of the system (x, y, and z).

The ADCS's structural model includes 42 equations including 42 unknowns variables, 15 known variables, and 9 faults that appear as parameters in the equations. The majority of these equations are based on three behavioral equations that address the system dynamics along the x, y, and z axes. The biadjacency matrix of the system is given by Figure 3. The links between relations and unknown variables are represented in blue, those with faults in red and those with known variables in black.

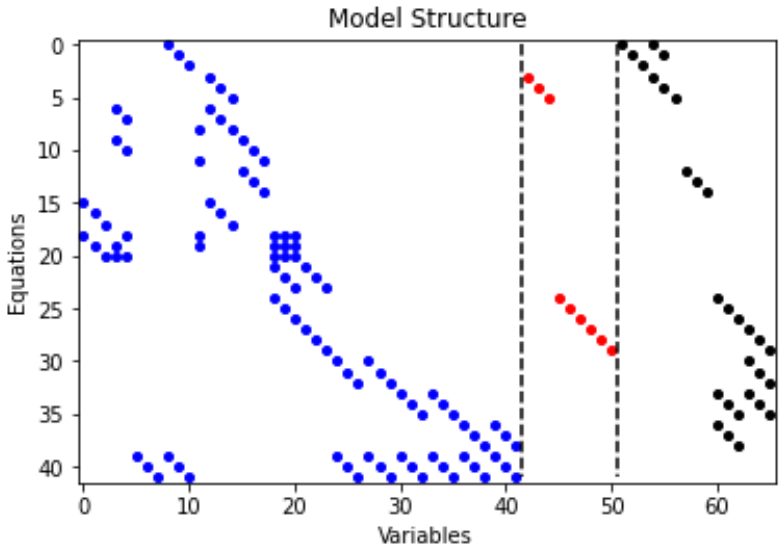


Figure 3 Biadjacency matrix of the ADCS system.

6.2 Results

Using the "Fault Diagnosis Toolbox"[2], 2448 MSO sets are found for this model, but only 9 MTES. There are also 2448 FMSO sets, as each MSO set has a non empty fault support.

In our example, we have $n_e = 42$, $n_f = 9$, and $\varphi = 9$. Applying Equation 3 results in a maximum of 511 nodes traversed during the search for MTES, whereas applying Equation 2 gives a maximum number of nodes exceeding 10^9. These values clearly illustrate the difference in complexity between the two algorithms, with the MSO algorithm requiring the traversal of far more nodes. However, these are only upper bounds, as in practice, the equations can be grouped into equivalence classes, significantly reducing the actual number of nodes traversed by the algorithm. Table 9 presents the computation time required to compute both the MSO sets and the MTES. It shows that the computation time for MSO sets is significantly higher than for MTES in large systems.

Table 9 Computation time of both algorithms.

Algorithm	Calculation time
MSO algorithm	0.534 s
MTES algorithm	0.239 s

It is clear that the 2448 MSO sets for this model are not all useful for the generation of residuals. MSO sets with empty fault supports are not interesting, as they do not contribute to fault driven residual generation. This is why we focus on FMSO sets, which contain faults and are thus more relevant for diagnosis. However, in this case, all MSO sets contain faults, making the use of FMSO sets less critical. There must still be some MSO sets with identical fault supports or non-minimal ones. The MTES of this model, along with the associated fault supports, are given in Table 10.

[2] `https://faultdiagnosistoolbox.github.io/`

Table 10 MTES and their fault support for the ADCS example.

	MTES	Fault support
$MTES_1$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{24}, e_{30}\}$	$\{fvs_z\}$
$MTES_2$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{23}, e_{29}\}$	$\{fvs_y\}$
$MTES_3$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{22}, e_{28}\}$	$\{fvs_x\}$
$MTES_4$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{27}\}$	$\{frs_z\}$
$MTES_5$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{26}\}$	$\{frs_y\}$
$MTES_6$	$\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{25}\}$	$\{frs_x\}$
$MTES_7$	$\{e_6, e_9, e_{12}, e_{15}\}$	$\{frw_z\}$
$MTES_8$	$\{e_5, e_8, e_{11}, e_{14}\}$	$\{frw_y\}$
$MTES_9$	$\{e_4, e_7, e_{10}, e_{13}\}$	$\{frw_x\}$

For this example, all these MTES are MSO sets. These are the MSO sets with the minimal fault supports, containing the largest number of equations. The other MSO sets are not necessary as they have fault supports that are unions of these, or they can be expanded with other equations without altering the fault support while still remaining MSO sets. These 9 MTES are necessary and sufficient to achieve maximal isolability as their fault support only includes one fault.

7 Conclusion

In this paper, a thorough examination and comparison of key concepts in model-based fault diagnosis is provided, specifically focusing on the structural analysis concepts named the Minimally Structurally Overdetermined (MSO) sets, Fault driven Minimally Structurally Overdetermined (FMSO) sets, and Minimal Test Equation Supports (MTES).

This article conducts a detailed analysis of the rules, applications and interrelationships of MSO sets, FMSO sets and MTESs, providing insights into their respective strengths and limitations. It then systematically examines the algorithms used to identify MSO and MTES sets, discussing their computational efficiency and impact on diagnosis capabilities. The final part of this work applies these concepts to a practical example, demonstrating their effectiveness in a real-world scenario. More specifically, the algorithms are applied to the Attitude Determination and Control System (ADCS) of a satellite in low Earth orbit, and the structural model, algorithmic complexity involved and the possibility of diagnosing faults are studied. This results highlight the practical usefulness of MTES in reducing the computational load while maintaining robust fault isolation capabilities.

References

1 Elodie Chanthery, Anna Sztyber, Louise Travé-Massuyès, and Carlos Gustavo Pérez-Zuñiga. Process decomposition and test selection for distributed fault diagnosis. In *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices: 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings 33*, pages 914–925. Springer, 2020. doi:10.1007/978-3-030-55789-8_78.

2 Elodie Chanthery, Louise Travé-Massuyès, and Saurabh Indra. Fault isolation on request based on decentralized residual generation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(5):598–610, 2015. doi:10.1109/TSMC.2015.2479192.

3 Marie-Odile Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004. doi:10.1109/TSMCB.2004.835010.

4 Andrew L Dulmage and Nathan S Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

5 Mattias Krysander, Jan Åslund, and Erik Frisk. A structural algorithm for finding testable sub-models and multiple fault isolability analysis. In *21st International Workshop on Principles of Diagnosis (DX-10), Portland, Oregon, USA*, pages 17–18, 2010.

6 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):197–206, 2007. doi:10.1109/TSMCA.2007.909555.

7 Carlos Gustavo Pérez, Louise Travé-Massuyès, Elodie Chanthery, and Javier Sotomayor. Decentralized diagnosis in a spacecraft attitude determination and control system. *Journal of Physics: Conference Series*, 659(1):012054, 2015.

8 Carlos Pérez-Zuñiga, Elodie Chanthery, Louise Travé-Massuyes, Javier Sotomayor, and Christian Artigues. Decentralized diagnosis via structural analysis and integer programming. *IFAC-PapersOnLine*, 51(24):168–175, 2018.

One-Class Classification and Cluster Ensembles for Anomaly Detection and Diagnosis in Multivariate Time Series Data

Adil Mukhtar
Institute of Software Technology, Graz University of Technology, Austria

Thomas Hirsch
Institute of Software Technology, Graz University of Technology, Austria

Gerald Schweiger
Vienna University of Technology, Austria

Abstract

Real-world automated systems such as building automation, power plants, and more have benefited from data-driven learning methodologies for anomaly detection and diagnosis. Typically, these methodologies heavily rely on prior knowledge related to abnormal operations, i.e., data points labeled as anomalies. However, in practice, such labelled data points are often unavailable which poses challenges in effective anomaly detection, particularly in diagnosis. In this paper, we propose One-class Classification Cluster ENsembles (OCCEN) anomaly detection and diagnosis approach for multivariate time series data. OCCEN utilizes one-class classification learning methods for anomaly detection followed by the decomposition of anomalies into multiple clusters. Then each cluster is treated as a binary classification problem and classifiers are trained to learn cluster representations. These trained models in combination with explainable AI models are used to generate a ranked list of diagnoses, i.e., features. Finally, we re-rank those features to account for temporal dependencies through the dynamic time-warping technique. The practical evaluation of OCCEN for air handling units (AHU) demonstrates its effectiveness in identifying faults. The framework consistently outperforms the baseline in fault diagnosis, as higher scores are observed for detection and diagnostic evaluation metrics, including F1 score, intersection over union, *HitRate@k*, and *RootCause@k*.

2012 ACM Subject Classification Computing methodologies → Machine learning algorithms; Computing methodologies → Artificial intelligence; Computing methodologies → Model development and analysis; Hardware → Fault models and test metrics

Keywords and phrases Anomaly Detection and Diagnosis, Machine Learning, Explainable AI, One-class Classification

Digital Object Identifier 10.4230/OASIcs.DX.2024.14

Funding The work described in this paper is funded by the ECom4Future project "FIWARE DRIVEN ENERGY COMMUNITIES FOR THE FUTURE" under contract number FFG: 903927.

1 Introduction

Automated systems such as building automation [24] and industrial plants [10] are equipped with numerous sensors and control signals. These sensors play a vital role in helping operators keep track of the system's condition by monitoring an array of multivariate time series data produced by many integrated components [15, 3]. A key aspect of this monitoring is the identification of abnormalities within the system, enabling operators to conduct diagnostic analyses to pinpoint malfunctioning components [12]. However, this process presents significant challenges in practical scenarios due to several factors [18]:

1. The system's complexity often limits clear insights into its workings, including how sensors, controllers, and actuators are operating.



© Adil Mukhtar, Thomas Hirsch, and Gerald Schweiger;
licensed under Creative Commons License CC-BY 4.0



2. Anomalies might not only result from components that are failing or exhibiting abnormal behavior but could also be due to a variety of operation modes.

As a result, it is important to have a methodology that not only detects these anomalies but also offers a list of potential diagnoses. Such a list might include the likelihood of failure in specific sensors or controllers, and indications of malfunctions, thereby enabling a more efficient response to system irregularities.

Over the years, the exponential development of large-scale monitored equipment has motivated researchers to propose a variety of data-driven automated anomaly detection and diagnosis methodologies [7, 1]. A majority of these techniques heavily rely on the prior knowledge of faulty and non-faulty operations called *supervised* learning. One of the significant limitations of this learning approach is the scarcity of anomalous data points in historical data, consequently, rendering such learning methods infeasible for real-world applications. On the other hand, *unsupervised* learning methods require no prior anomalous data points and heuristically extract distinctive operation patterns within the historical data. Despite their effectiveness, these techniques may not be well suited for effective anomaly detection and diagnosis due to the following reasons:

- Most proposed methodologies are designed to target specific types of faults and rely on the physical characteristics of the system, resulting in limited generalizability and scalability [15].
- The prior knowledge of faulty operations in the data, i.e., target labels, is usually not available in real-world scenarios, therefore, supervised learning techniques are not considered feasible [8].
- Unsupervised clustering methods [12], including k-means clustering, hierarchical clustering, and density-based clustering, create different groups based on feature similarities. Yet, these techniques do not reveal the complex, non-linear interrelations among features, which can lead to suboptimal performance in diagnostic performance.
- In real-world scenarios, it is desirable to assist operators with a ranked list of diagnoses. However, existing methods fall short of providing a nuanced, ranked list of diagnoses that aligns with the operators' requirements.
- Finally, the adaptability of the existing methods is usually limited due to their reliance on specific data structures, predefined models, and the need for extensive labeled datasets in supervised learning, or the lack of interpretability in unsupervised learning.

In this paper, we tackle the previously mentioned challenges and propose One-class Classification Cluster Ensembles (OCCEN) technique that orchestrate already existing unsupervised and supervised learning methods for anomaly detection and diagnosis. Specifically, OCCEN utilizes a one-class classification approach to pinpoint anomalous data points, an approach that does not require priors. Following the anomaly detection task, we employ clustering techniques to group these anomalies into various groups. Subsequently, each cluster is treated as a separate binary classification learning problem to uncover the non-linear relationships among features within each cluster and further learn distinct features of clusters. The intuition is that partitioning anomalies into clusters will enable the learning model to extract stable patterns among anomalous data points. To generate a ranked list of diagnoses, i.e., the impact of specific features on the detected anomaly, cluster assignments are treated as input to the explainable AI models such as Local Interpretable Model-Agnostic (LIME) and Shapley Additive Explanations (SHAP). These models identify and rank the most influential features, or diagnoses, for each instance during a specific time step. Once the features are identified for a certain number of time steps, i.e., m, we measure the distance of each ranked feature through *Fast Dynamic Time Warping* (FastDTW) [28], which measures the

distance and accounts for temporal dependencies in time series data. In simpler terms, the main motivation behind this step is to rank the most relevant features, i.e., diagnosis, of the cluster assignments. This additional step in the ranking process allows for a more nuanced understanding of how certain features influence anomalies over time. Through this comprehensive approach, we aim to create a more effective and contextually aware system for anomaly detection and diagnosis, leveraging both unsupervised and supervised learning techniques.

We evaluated our approach on the time series dataset published by Granderson et al. [11]. The simulated dataset contains minute recordings of the Air Handling Unit (AHU) sensors, capturing both their faulty and normal operational states. For the task of fault detection, we considered fault-free recordings, i.e., normal operations, as input to the one-class classification methods and the evaluation with 10-fold cross-validation shows that Elliptic Envelope performs well and identifies faults of all types with average metric scores of 89.4% F1, 94.2% precision, and 92.6% recall. As for the fault diagnosis, we rely on k-means clustering technique to group predicted faults and evaluate various binary classification methods on each cluster. We found that the Random Forest classifier performs well, on average, across all evaluation metrics. Finally, for the baseline comparison in diagnosing faults, we evaluated our method against a standard approach that combines the one-class classification technique with explainable models, measuring how well each method identifies and ranks features (diagnoses). We found that our method significantly outperforms the standard approach in terms various diagnosis metrics, i.e., Intersection over Union (IoU), HitRate@k, and RootCause@k. In summary, we contribute the following with our work:

- OCCEN combines learning aspects of both supervised and unsupervised techniques for improved performance. Unlike existing methods, which rely primarily on labeled faulty instances for fault diagnosis, our methodology relies only on fault-free time series data.
- We introduce a novel methodology of combining one-class classification with clustering techniques incorporating temporal dependencies to boost unsupervised fault diagnosis performance. Furthermore, we provide a ranked list of diagnoses to assist the operator in effectively diagnosing faults. To the best of our knowledge, our methodology is the first that considers combining learning methods in this way for anomaly detection and diagnosis in time series data.
- We evaluated OCCEN on the simulated data for real-world application of detecting and diagnosing faults in air handling units. The evaluation suggests that the proposed methodology consistently achieves better performance compared to the baseline.

In the following, we first discuss relevant works in Section 2. We formally define the problem statement and describe the building blocks of our proposed framework (OCCEN) in Section 3. The experimental setup details are provided in Section 4 followed by the discussion on obtained results in Section 5. We then discuss the potential implications and possible limitations of our work in Section 6, and finally conclude this work in Section 7.

2 Related Work

A variety of data-driven methodologies have been proposed over the years for the task of anomaly detection. Typical methods from clustering and classification types are considered the most relevant techniques for the task. For example, the k-means clustering technique [23] is usually adapted to group anomalous and normal data points into various clusters through similarity and/or distance measures [16, 19]. In this context, Li et al. [19] proposed a framework that analyzes long multivariate time series by dividing them into shorter subsequences

using a sliding window. Enhanced Fuzzy C-Means (FCM) clustering and particle swarm optimization are applied to identify structures and assign anomaly scores. Additionally, it includes a shape anomaly detection step, utilizing autocorrelation to capture shape information and address time shifts. In another work, Liu et al. [21] proposed an anomaly detection method for condition monitoring data, utilizing auxiliary feature vectors and DBSCAN clustering to differentiate between valid and invalid anomalies. This method segments the data based on interruptions, constructs auxiliary feature vectors for each segment, and applies Density-Based Spatial Clustering (DBSCAN) for accurate pattern recognition. A heuristic based on the 'number of clusters–Eps' curve is proposed to optimize DBSCAN parameters, effectively identifying normal patterns and anomalies in unlabelled, imbalanced datasets with rare anomalies. Clustering methods, while effective and not needing labeled data, are incapable of capturing temporal dependencies in multivariate time series across different time steps [31], a crucial aspect in anomaly detection and diagnosis. In our work, we address this issue by integrating temporal dependencies into the framework by combining explainable AI function output with the dynamic time-warping technique for the diagnostic process.

In addition to clustering techniques, classification methods are also applied [13, 6, 5]. Fouzi et al. [13] presented an effective approach for fault detection in Photovoltaics (PV) arrays, merging model-based strategies with One-class SVM (One-SVM) clustering. It utilizes a simulation model to replicate normal PV array behavior, applying One-SVM to the resulting discrepancies for fault identification. This approach, particularly adept at handling nonlinear features, showed superior performance in fault detection compared to other clustering methods in tests on a 9.54 kWp grid-connected PV plant. In contrast, this work proposes to combine unsupervised and OCC data-driven methods for anomaly detection and diagnosis. In another work, Beghi et al [5] proposed a novel semi-supervised, data-driven method. This technique utilizes Principal Component Analysis (PCA) to differentiate anomalies from typical operational variations, coupled with a reconstruction-based contribution method to identify fault-related variables. Fault diagnosis is facilitated through a decision table linking the impact of faults to specific features. The effectiveness of the Fault Detection and Diagnosis (FDD) algorithm is evaluated using experimental datasets from two different water chiller systems.

Finally, advanced methods such as deep learning are also used in the literature [31, 9]. Zhang et al. [31] proposed the Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) approach for detecting and diagnosing anomalies in multivariate time series data. It generates multi-resolution signature matrices to encapsulate different system status levels over time. This method effectively integrates a convolutional encoder and an attention-based ConvLSTM network to capture inter-sensor correlations and temporal dynamics, using a convolutional decoder to reconstruct and analyze these matrices for anomaly identification. In another work, Garg et al. [9] reviewed plenty of deep learning methods and evaluated the performance based on the newly proposed metric *composite F-score* (F_{c1}).

To summarize, we briefly discussed the adaptation of data-driven methodologies for anomaly detection and diagnosis in time series data. While these techniques share common characteristics such as feature generation, transformation, sampling, etc., the modeling is performed through one particular type of learning schema, i.e., clustering or classification. On the other hand, clustering methods are often preferable for real-world cases, however, such methods have limitations, i.e., the inability to capture temporal dependencies and non-linear relations among features. Therefore, in this work, we propose to combine classification and clustering methods and apply explainable AI in conjunction with the dynamic time-warping technique in a framework.

3 Framework & Methodology

In this section, we describe the problem formulation in detail along with the details regarding the building blocks of our proposed methodology.

Problem Statement: Let $T = \{t_1, t_2, t_3, ..., t_m\}$ represents time series data observations for fault-free operations. For each data point $t_i \in T$, there is k number of recorded features, i.e., multivariate, hence, $t_i = \{x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,k}\}$.

3.1 Anomaly Detection and Diagnosis

The overall framework of OCCEN is presented in Figure 1. For the task of anomaly detection, the objective is to learn a function $f : T \rightarrow \{0, 1\}$ where $f(t_i) = 0$ represents a non-anomalous instance and $f(t_i) = 1$ indicates anomalous data point. Furthermore, it is assumed that characteristics of T represent normal operations, and deviations from normal patterns are indicative of faults.

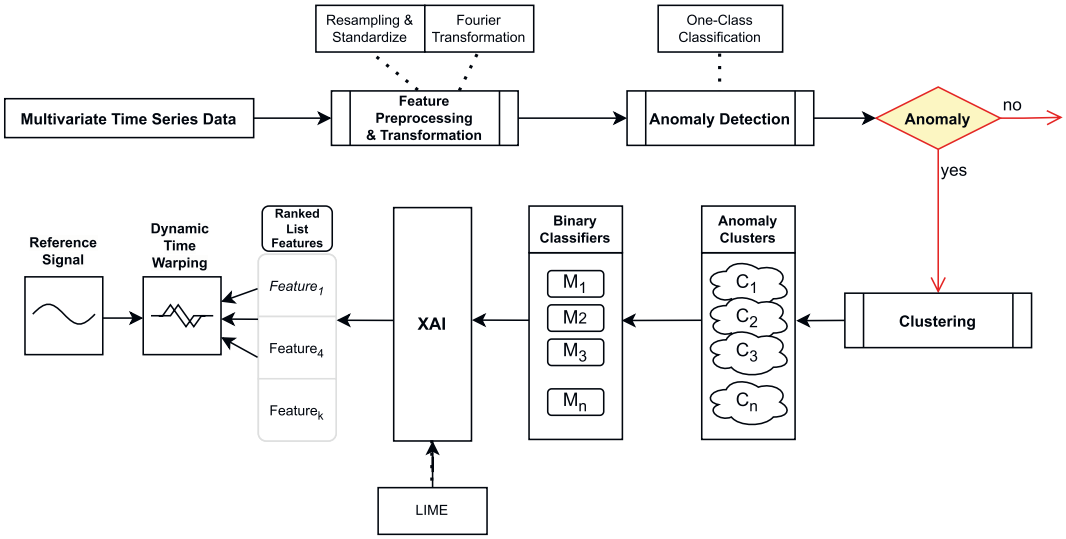


Figure 1 OCCEN Anomaly Detection and Diagnosis Framework.

The design of the diagnosis phase is provided in Algorithm 1. We now describe the algorithm in more detail. As part of the diagnosis, let $F = \{t_i | f(t_i) = 1, t_i \in T\}$ be the set of all the detected anomalies. Then clustering C can be applied to F such that $C(F) = \{c_1, c_2, c_3, ..., c_n\}$ where n indicates the number of clusters obtained.

For each cluster, train a binary classifier B_j that distinguishes instances among clusters by learning distinctive cluster patterns. To train classifier B_j, let D_{c_j} be the cluster instances in c_j, then randomly select an equal number of instances from $\bigcup_{i \neq j} D_{c_j}$, i.e., the union of instances from other clusters to form D_{other}. Then the training set for B_j is $D_{c_j} \bigcup D_{other}$ ensuring a balanced representation of the target cluster and other clusters.

To generate a ranked list of diagnoses, let E be the explainable model, such as LIME, applied to each instance $t_i \in D_{c_j}$ such that $E(t_i)$ provides explanations, ranked list of features, for the cluster assignment. Then identify top-ranked features $F_{top}(t_i)$ that contribute to the assignment of t_i. Retain the top-ranked features $F_{top}(t_i)$ from $E(t_i)$ over a sequence of length m window. For each feature $f_r \in F_{top}(t_i)$ calculate $FastDTW$ distance between f_r in the faulty sequence and its counterpart sequence in the fault-free observations T_{normal} and denote

Algorithm 1 Diagnosis through Anomaly Detection, Clustering, and Feature Ranking.

Require: T: Set of all instances (both normal and anomalous)
Require: $f(t_i)$: Anomaly detection function (1 if anomalous, 0 if normal)
Require: E: Explainable model (e.g., LIME)
Require: m: Window length for feature retention
Require: T_{normal}: Set of fault-free (normal) observations
Ensure: Ranked list of critical features for root cause diagnosis

Step 1: Anomaly Detection
$F \leftarrow \{\}$ ▷ Set of all detected anomalies
for each $t_i \in T$ **do**
  if $f(t_i) = 1$ **then**
    $F \leftarrow F \cup \{t_i\}$
  end if
end for

Step 2: Clustering
Apply clustering C on F: $C(F) = \{c_1, c_2, \ldots, c_n\}$

Step 3: Train Binary Classifiers for Each Cluster
for each cluster $c_j \in C(F)$ **do**
  $D_{c_j} \leftarrow$ instances in c_j
  $D_{\text{other}} \leftarrow$ randomly select $|D_{c_j}|$ instances from all other clusters $i \neq j$
  $D_{\text{train}} \leftarrow D_{c_j} \cup D_{\text{other}}$
  Train binary classifier B_j on D_{train}
end for

Step 4: Feature Explanation and Ranking
for each cluster $c_j \in C(F)$ **do**
  for each instance $t_i \in D_{c_j}$ **do**
    $F_{\text{top}}(t_i) \leftarrow$ top-ranked features from $E(t_i)$
    Retain $F_{\text{top}}(t_i)$ over a window of length m
  end for
end for

Step 5: Calculate FastDTW Distance for Top Features
for each feature $f_r \in F_{\text{top}}(t_i)$ **do**
  $DTW_{\text{distance}}(f_r, T_{\text{normal}}) \leftarrow \text{FastDTW}(f_r \text{ in faulty sequence}, f_r \text{ in } T_{\text{normal}})$
end for

Step 6: Rank Features by FastDTW Distance
Rank features in $F_{\text{top}}(t_i)$ based on $FastDTW_{\text{distance}}(f_r, T_{\text{normal}})$ in descending order

Step 7: Apply Truncation
Truncate the ranked feature list to retain only the most critical features with the highest FastDTW distances
return Truncated list of critical features for root cause diagnosis

this as distance $FastDTW(f_r, T_{normal})$. Now rank the features based on the $FastDTW$ distance in descending order to apply truncation. In the context of diagnoses, these features are considered critical for identifying the root cause based on the severity [31], i.e., distance, as these features show the most deviation from normal behavior. In the following sections, we now describe the learning methods employed and how these are adapted in this work.

3.2 One-class Classification (OCC)

Labeling large datasets, particularly high-resolution time series data, is very challenging in real-world situations because it requires a lot of time and effort. The sparsity of class labels (minority class), e.g., anomalies in our case, in the datasets negatively affects the learning capability and performance of the model due to the learning bias towards the dominating class labels (majority class). To address this, one-class classification (OCC) methods are often used [17]. One-class classification methods are a special case of binary- or multi-classification learning methods. These methods are extensively used for detecting anomalies and novelties in time series data [2]. The main concept involves learning the distribution of a single class, typically normal observations, and establishing decision boundaries that differentiate between inliers and outliers. Formally, boundaries for object z, i.e., each object z in the dataset is a multivariate characterization of non-anomalous characteristics, can be defined based on the following concepts [4]:

- the measure of how far away, i.e., $d(z)$, or how likely it is, i.e., $p(z)$, (similarity) that an object z resembles the target class, which is represented by a training set X_{train}.
- a threshold, either θ_d or θ_p, applied to this distance or probability (similarity).

The similarity and distance measurements are used as the decision-making function for categorizing new, unknown objects as either inliers or outliers. For example, an object is identified as an outlier if $d(z) > \theta_d$ or $p(z) < \theta_p$, and the opposite criteria apply for classifying it as an inlier. A variety of OCC methods proposed in the literature [17] utilize either one or both of the key concepts of similarity and distance analysis. For instance, OneClassSVM [29] defines boundaries by mapping objects into higher dimensional feature space to find hyperplanes characterizing normal data points. Whereas, the Local Outlier Factor (LOF) assesses the local density deviation of a given data point concerning its neighbors, identifying anomalies based on significant differences in local densities. Nonetheless, other methods like Elliptic Envelope, Isolation Forest [20], Gaussian Mixture Models (GMM) [26], and advanced methods such as auto encoders [25] primarily learn the normal representation of the given objects. Each of these methodologies applies the concepts of distance and similarity in different ways, tailored to identify anomalies or outliers in various types of data effectively. Their effectiveness can depend on the specific characteristics of the data and the context of the problem being addressed.

In our study, we utilize One-Class Classification (OCC) methods to identify anomalies within multivariate time series data, previously denoted as $T = \{t_1, t_2, t_3, ..., t_m\}$. Each object in T is presumed to be normal and is used as input for the training of the model. The primary objective of the model is to accurately learn the characteristics of normal, anomaly-free data points. This learning is then applied to classify new data points as either normal or anomalous. The effectiveness of the model is measured by its ability to accurately classify both anomalous and non-anomalous new data points, with a particular focus on achieving high precision and recall in this classification process. Essentially, the model's success relies on its proficiency in correctly identifying anomalies while minimizing false positives and negatives.

3.3 Learning Clusters Representations

Diagnosis in non-linear dynamic systems proves difficult without prior information, particularly when anomaly detection is trivial, i.e., anomalies appear as clear outliers or sudden signal disruptions. Therefore, in this step, we design a diagnostic analysis framework that first groups anomalous objects based on their characteristics and subsequently learns each group's pattern representation. The intuition behind this methodological design is that similar faulty working conditions exhibit common anomalous characteristics, by extracting the representation of these akin faults, we can improve the association of features within each faulty condition.

In our approach to diagnosis, we initially use clustering techniques to group objects classified as anomalies by the OCC technique. These clusters capture the common characteristics of the anomalous objects, resulting in better feature associations within each cluster. However, clustering primarily groups objects based on similarities or density measures [12], and does not inherently extract complex, non-linear relationships within each cluster. To address this, we improve the cluster representations by assigning a unique label to each cluster and treating each cluster as a separate binary classification problem. This step is key in extracting inter-cluster characteristics. By doing so, the binary classifier aids in extracting more nuanced and distinct patterns from each cluster, thus providing a more precise understanding of each cluster. To achieve this, we train a separate binary classifier on each cluster. The learning data for the classifier is generated by selecting the objects of the relevant cluster and assigning the label 0 and randomly selecting an equal number of objects from other clusters, i.e., label 1, to account for the balanced representation of both classes. This process is repeated for all the clusters generated by the clustering technique.

3.4 Explainable AI

Explainable AI methods such as LIME [27] and SHAP [22] are often considered useful to explain the decision made by the classifier [14]. This is accomplished by generating a ranked list of features that are indicative of the decision. These features are considered the most influential in terms of decision-making of the underlying base model. In this work, we use explainable methods as the first step toward the task of feature ranking for the anomalies. We identify the most significant features for each anomalous object in the time series. This is done by assigning the anomalous object to a particular cluster and then using the relevant binary classifier associated with that cluster, as the base model for the explainable AI method. So far, feature generation occurs at a specific time step t, and time series data often includes temporal dependencies that can impact the effectiveness of fault diagnosis. To address this, we take into account a set of features generated by explainable methods over a time sequence spanning a length of m. However, a longer time sequence will result in a large number of diagnoses, i.e., features, and it may affect the fault diagnosis performance. Therefore, to further pinpoint the diagnoses, we rely on the FastDTW to rank the features based on point-wise distance.

3.5 Fast Dynamic Time Warping (FastDTW)

FastDTW is a well-known technique to compare the similarity/distance between two given temporal time series [28]. It relies on calculating the similarity across various phases by minimizing the shifts and time distortion by "elastic" transformation. FastDTW performs better in precisely calculating and comparing two signals, particularly in time series data. Its capability to adjust for variations in time and phase differences allows for a more accurate

alignment and comparison of sequences than traditional methods [30]. This makes FastDTW an optimal choice for accurately determining the similarity or dissimilarity between two signals, especially when dealing with temporal discrepancies. For example, as shown in Figure 2, the analysis of metrics calculated via FastDTW versus Euclidean distance for both normal and faulty temperature sensor signal operations indicates better FastDTW's efficacy. In particular, FastDTW distance represents the total cost (distance) of aligning the two sequences (faulty and fault-free) through FastDTW. We note that FastDTW provides a more precise and reliable measure for detecting discrepancies in the sensor's readings, especially in distinguishing between normal and abnormal operational conditions. Therefore, we utilize FastDTW to determine the distance between features identified by the explainable method over m time steps and the corresponding normal observations of these features during the same period in a fault-free state. To achieve this, we retain the features identified by the explainable method across m time steps and then re-rank them by computing the distance between each feature and its corresponding normal state observation when the system is free of faults. Finally, a list of diagnoses is produced, ordered by their distances, with the diagnosis having the greatest distance placed at the top of the list.

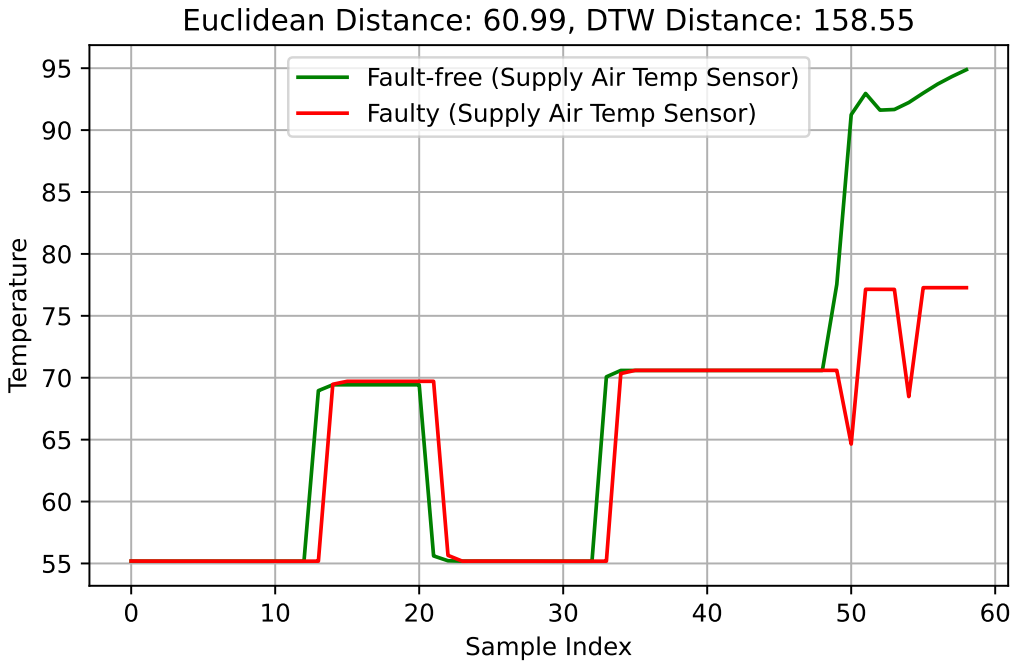


Figure 2 FastDTW vs Euclidean Distance Analysis.

4 Experimental Setup

4.1 Data

In this study, we employ a simulated dataset of a single-duct air handling unit (AHU), published by Granderson et al. [11]. The dataset comprises one year of operational data, incorporating both faulty and nominal system behaviors, with a total of 525,600 time samples representing each operational condition. Faults are imposed on sensors and control sequences at various biases (see Table 1), but our diagnostic approach identifies fault types without

focusing on bias levels. We train OCC models for anomaly detection using only normal operations data. Pre-processing includes resampling to hourly averages, applying Fast Fourier Transformation (FFT), and feature scaling with the min-max method. These steps, along with trained models, are applied to preprocess test data, ensuring no data leakage.

Table 1 Summary of Fault Imposition Methods for AHU, adapted from [11].

Fault Type	Fault Annotation	Method of Fault Imposition
Supply air temperature sensor bias	Sa_temp	Add a bias value to the sensor output
Outdoor air temperature sensor bias	*Oa_temp*	Add a bias value to the sensor output
OA damper stuck	*Dmpr_stk*	Automated override of outdoor air damper position to indicate it is stuck
Cooling coil valve leaking	*Cvlv_lkg*	Adjusted the coil valve position value when the control signal is zero
Cooling coil valve stuck	*Cvlv_stk*	Override of the coil valve position to indicate that the valve is stuck

4.2 Baseline

We evaluated the efficacy of different OCC methods in detecting anomalies and considered the best-performing method for the fault diagnosis analysis. To assess fault diagnosis performance, we established a simple baseline by employing explainable AI models immediately following the OCC methods, unlike OCCEN. The explainable model generates a list of ranked features, i.e., diagnoses, for a given time step. The top features ranked by the XAI model are then retained for over m time steps and further re-ranked based on distance measures calculated by FastDTW. The design of the baseline approach is shown in Figure 3. It is based on the premise that explainable AI models are generally good at identifying the key features that affect the decisions of the base model [27, 22] and FastDTW technique will further improve the ranking. The objective is now to assess how well this baseline performs in feature ranking compared to OCCEN.

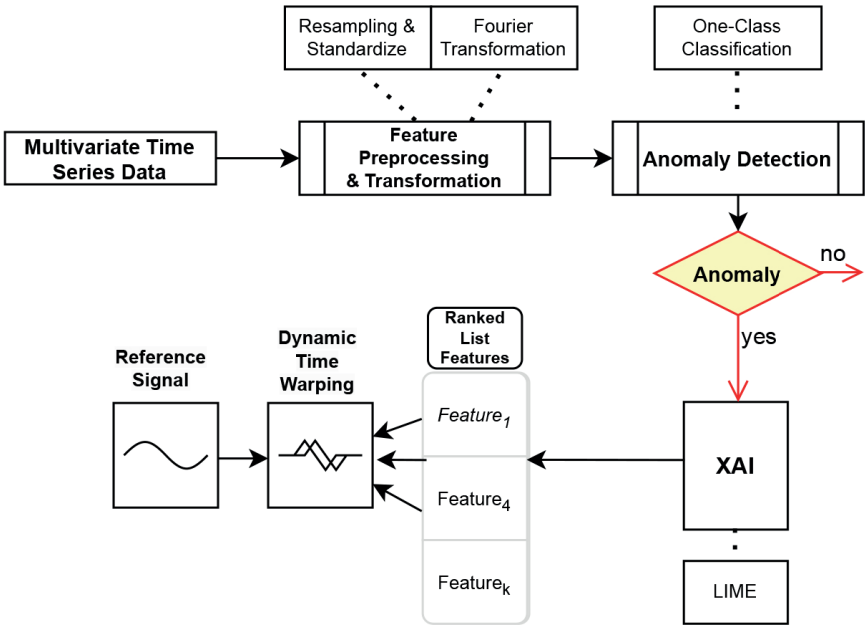


Figure 3 Baseline Approach Design.

4.3 Evaluation Procedure & Metrics

We experimented with multiple OCC methods for the classification of anomalies. We partitioned the dataset into two splits, using mid-August to December (30%) as a hold-out set and January to mid-August (70%) for time-series cross-validation in the anomaly detection task. Unlike traditional cross-validation, time-series cross-validation maintains the temporal order of observations, crucial for time-series data. We employed sequential 10-fold splitting, ensuring the training set includes only past data relative to the test set to prevent data leakage. For performance evaluation, we used Precision, Recall, and F1 Score, reporting the mean scores averaged over the folds for the OCC methods in Section 5. After identifying the most effective model, we used it to predict anomalies in the faulty dataset (January to mid-August) and applied various clustering methods, finding k-means most effective with an optimal cluster number of 4. This number is based on elbow method and sillhouette coefficient analysis. We tested different classification models to understand each cluster's unique characteristics. The unseen hold-out test set was pre-processed and fed into our anomaly detection approach (OCCEN). Detected anomalies were categorized using k-means clustering, and corresponding trained binary classifiers with explainable models (LIME and SHAP) were applied. LIME was more efficient for explanation generation, while SHAP was more computationally intensive. We experimented with different sliding window sizes, focusing on LIME-prioritized features and using FastDTW to rank features based on their distance from normal signals.

In our study, each fault type has a single root cause, but in non-linear dynamic systems, one failure can impact multiple components, making it crucial to examine these effects to identify the root cause. We consulted building automation experts to review the dataset and list possible diagnoses for each fault type. To evaluate fault diagnosis performance, we adapted two metrics: *Overlap@P* and *HitRate@k*. *Overlap@P* measures the overlap between true diagnoses and the top $P \times |GT|$ ranked diagnoses, with P as 150% or 200% and $|GT|$ as the number of true diagnoses. Most fault types have 5 true diagnoses, except *Sa_temp* with 2, capping ranked diagnoses at 10 when P is 200%. We also used *HitRate@k*, which checks if at least one true diagnosis is within the top k ranked diagnoses. Our dataset includes five fault variants: *Sa_temp*, *Oa_temp*, *Dmpr_stk*, *Cvlv_lkg*, and *Cvlv_stk*. To measure the accuracy of identifying root causes, we introduced *RootCause@k*, assessing the proportion of instances where the true cause is ranked among the top k diagnoses. We compared the effectiveness of the baseline and OCCEN using these metrics across sliding window sizes from 3 to 24 for diagnosis performance.

5 Results & Analysis

In this section, we discuss and report the results of anomaly detection and diagnosis for the experimental setup.

5.1 Anomaly Detection

The average results of the anomaly detection task are presented in Figure 4. The Elliptic Envelope model consistently shows high precision and recall across all fault types, resulting in strong F1 scores, particularly in detecting Cooling Coil Valve Stuck (*Cvlv_stk*) anomalies. Its precision above 93% and recall above 90% in most cases reflect its robustness and reliability in anomaly detection. The Isolation Forest model, however, presents a significant drop in performance, especially in recall values, which are notably low across all fault types. The

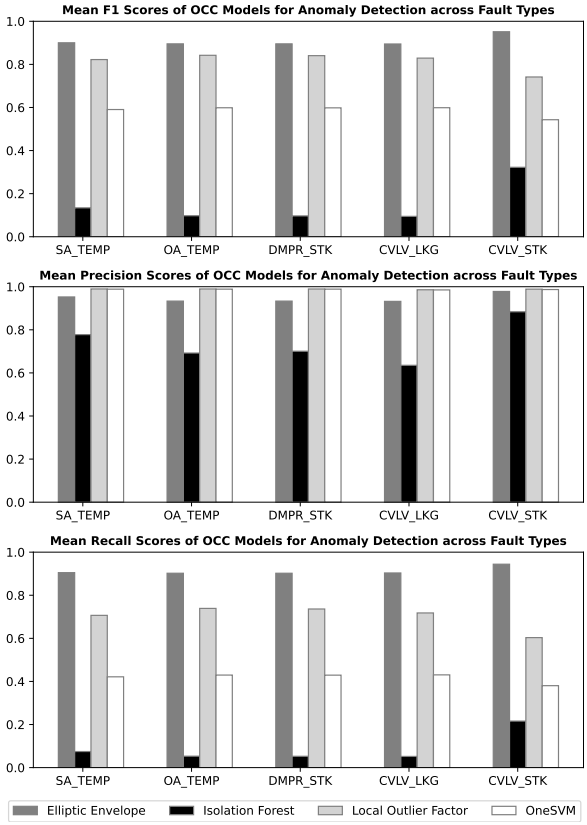


Figure 4 10-Fold Cross Validation Evaluation of OCC Methods.

Isolation Forest shows a precision of over 63% for most faults but struggles with recall, dipping as low as 5% in cases like Cooling Coil Valve Leakage (*Cvlv_lkg*) and Outdoor Air Damper Stuck (*Dmpr_stk*). The Local Outlier Factor (LOF) model exhibits high precision across all fault types, similar to the Elliptic Envelope model, but has a moderately lower recall, especially in the case of Cooling Coil Stuck (*Cvlv_stk*). Despite this, the LOF model maintains decent F1 scores, indicating a balanced trade-off between precision and recall. The One-Class SVM (OneSVM) model shows exceptional precision but considerably lower recall across all faults. This disparity results in moderate F1 scores, which are notably lower than those of the Elliptic Envelope and LOF models. The OneSVM is particularly weak in recall for Cooling Coil Stuck (*Cvlv_stk*), leading to its lowest F1 score among the fault types. Overall, the Elliptic Envelope model demonstrates the most balanced and effective performance in anomaly detection across various fault types, with the Local Outlier Factor (LOF) model following closely. In contrast, the Isolation Forest and OneSVM models, despite their high precision in some cases, lag in overall effectiveness due to significantly lower recall values.

5.2 Clusters Learners Performance

Analyzing the results in Table 2, we assess the performance of binary classifiers trained on each cluster using 10-fold cross-validation. High precision values indicate the model's ability to correctly identify anomalies within specific clusters, confirming the relevance and discriminative power of the features used. High recall values show the model's effectiveness in

identifying true positives, ensuring critical anomaly characteristics are captured. Consistently high F1 scores suggest a balance between precision and recall, indicating robust and reliable anomaly indicators within each cluster. Comparing model performances, Random Forest and Decision Tree models showed consistently high performance across all clusters, especially in Clusters 0 and 1, with F1 scores reaching 0.99. The Support Vector Machine also performed well, maintaining an F1 score of 0.97 across Clusters 0, 1, and 2. However, the Naive Bayes classifier showed variability, particularly in Cluster 2, where it had lower recall (0.68) and F1 Score (0.80), indicating issues with false negatives.

Since clusters are formed based on similarities in the data, each cluster likely represents different operational states or types of faults. The variation in model performance thus reinforces the idea that different clusters indeed capture distinct aspects of the system's operation, which can be utilized for more accurate fault diagnosis. The relatively low standard deviations in the metrics imply that the models' performances are consistent across different folds of the cross-validation. This consistency is important for practical applications as it indicates the models' robustness and reliability. The process of learning patterns within each cluster allows for the capture of cluster-specific patterns, which can help in better fault diagnosis. For this work, we selected the Random Forest binary classifier as our primary clusters learner as it outperforms other classifiers for most of the clusters.

Table 2 10-Fold cross validation Mean (Std) scores for the classification models learned on each cluster.

Model	Cluster Number 0			Cluster Number 1			Cluster Number 2			Cluster Number 3		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Decision Tree	0.96 (0.08)	0.99 (0.00)	**0.97** (0.05)	0.99 (0.00)	0.99 (0.00)	**0.99** (0.00)	0.96 (0.11)	0.99 (0.00)	0.97 (0.06)	0.99 (0.01)	0.99 (0.00)	**0.99** (0.00)
Naive Bayes	0.95 (0.08)	0.99 (0.00)	**0.97** (0.05)	0.93 (0.13)	0.99 (0.00)	0.96 (0.08)	0.99 (0.02)	0.68 (0.09)	0.80 (0.05)	0.96 (0.03)	0.99 (0.00)	0.98 (0.01)
Support Vector Machine	0.96 (0.08)	0.99 (0.00)	**0.97** (0.05)	0.96 (0.10)	0.99 (0.00)	0.97 (0.06)	0.96 (0.10)	0.99 (0.00)	0.97 (0.06)	0.99 (0.00)	0.99 (0.00)	**0.99** (0.00)
Random Forest	0.96 (0.08)	0.99 (0.00)	**0.97** (0.05)	0.99 (0.00)	0.99 (0.00)	**0.99** (0.00)	0.92 (0.14)	0.99 (0.00)	0.95 (0.08)	0.99 (0.00)	0.99 (0.00)	**0.99** (0.00)
Logistic Regression	0.96 (0.09)	0.97 (0.01)	**0.97** (0.05)	0.94 (0.10)	0.99 (0.00)	0.96 (0.06)	0.96 (0.09)	0.99 (0.00)	**0.98** (0.05)	0.98 (0.02)	0.99 (0.00)	0.98 (0.01)

5.3 Diagnostic Performance

In the following, we provide a detailed analysis and discussion on the diagnostic performance OCCEN compared to the baseline across different types of faults.

5.3.1 Overlap@P

The baseline and OCCEN comparative analysis is presented in Figure 5. Overall, OCCEN consistently performs better than the baseline at ranking relevant diagnoses for all fault types and achieves higher overlap percentages. This becomes more pronounced for larger window sizes.

The diagnostic results for the fault type *Sa_temp* show that OCCEN consistently outperforms the baseline across all window sizes. The overlap metrics at 150% and 200% increase substantially with OCCEN as the window size grows, compared to the baseline performance. This indicates that larger window sizes provide more temporal information for diagnosing *Sa_temp* faults with OCCEN. Despite *Sa_temp* being a challenging fault to diagnose due to having only two true diagnoses amidst more ranked diagnoses, OCCEN

shows significant improvement over the baseline. For the *Oa_temp* fault type, OCCEN also outperforms the baseline consistently, although there is a slight decrease in overlap beyond a window size of 9, suggesting a potential saturation point. Nonetheless, OCCEN achieves 80% overlap with true diagnoses at 200% compared to only maximum of 49% for the baseline. The *Dmpr_stk* fault type shows significant performance improvement with OCCEN. The overlapping score is up to 82% and 92%, compared to the baseline's maximum of 53% and 67%. This indicates OCCEN's efficacy in diagnosing *Dmpr_stk* faults, particularly with larger window sizes. Similarly, for the *Cvlv_lkg* fault type, OCCEN consistently outperforms the baseline. The higher overlaps across all window sizes, though the increase with window size is less pronounced, suggesting smaller windows may suffice for this fault type. OCCEN achieves overlaps of up to 64% at 150% and 76% at 200%, significantly higher than the baseline. *Cvlv_stk* fault type also shows OCCEN achieving higher scores compared to the baseline across all window sizes. The overlaps increase with window size for OCCEN, reaching up to 62% at 150% and 77% at 200%, whereas the baseline reaches a maximum of 43% and 54%, respectively.

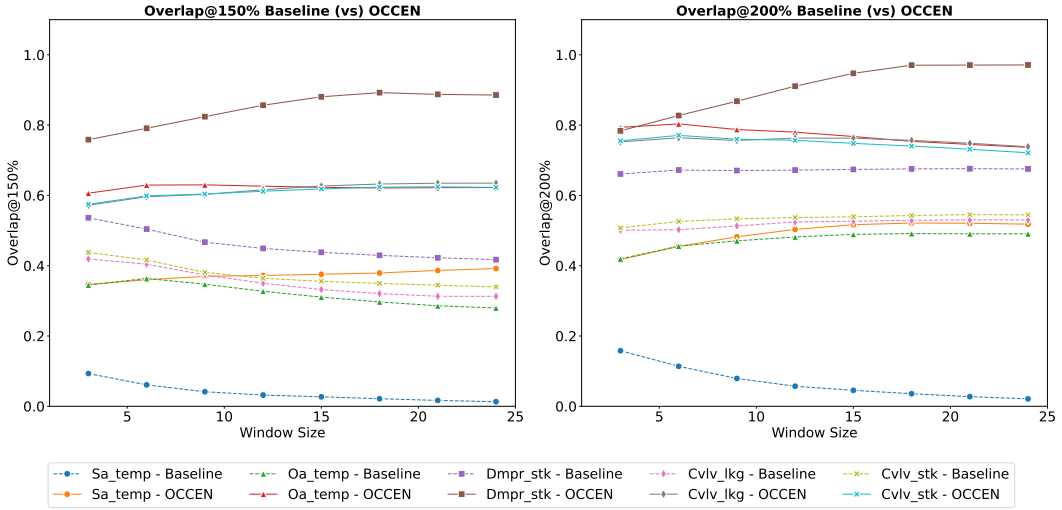


Figure 5 Mean score comparison of Overlap@P metric for baseline and OCCEN.

Overall, OCCEN demonstrates significant improvements over the baseline across all fault types, consistently achieving higher overlap metrics. The varying impacts of window size across different fault types suggest that while some faults benefit from larger windows, others like *Cvlv_lkg* perform well with smaller windows. Nevertheless, OCCEN's overall performance highlights its effectiveness and robustness in diagnosing faults.

5.3.2 HitRate@k

The overall evaluation of HitRate@k metric is presented in Figure 6. Overall, the results demonstrate clear performance improvements for OCCEN in identifying at least one true diagnoses in top k ranked diagnoses and it improves as window size increases. Conversely, the baseline shows, in general, lower hit rate scores but performs relatively better for large window size.

The fault type *Sa_temp* is challenging to diagnose due to having only two true diagnoses, while the ground truth diagnoses exceed this number. Despite this complexity, OCCEN demonstrates a significant improvement over the baseline across all window sizes. The hit

rates for k = 3, 5, and 7 steadily increase for OCCEN. In contrast, the baseline method struggles, with hit rates peaking at 18%, 45%, and 75% for the same window size. For the fault type *Oa_temp*, OCCEN outperforms the baseline across all window sizes. The hit rates for OCCEN are significantly higher across all k-values. The baseline method, however, exhibits lower hit rates, with a maximum of 88% at a window size of 3 for k=7. The results for the fault type *Dmpr_stk* reveal a similar trend, with OCCEN consistently achieving higher hit rates than the baseline. OCCEN maintains near-perfect hit rates across all k-values and window sizes, with hit rates reaching up to 100% at larger window sizes. The baseline, on the other hand, peaks at 34%, 75%, and 98% for k = 3, 5, and 7, respectively. These results highlight the better identification of true diagnosis by the OCCEN for *Dmpr_stk* faults.

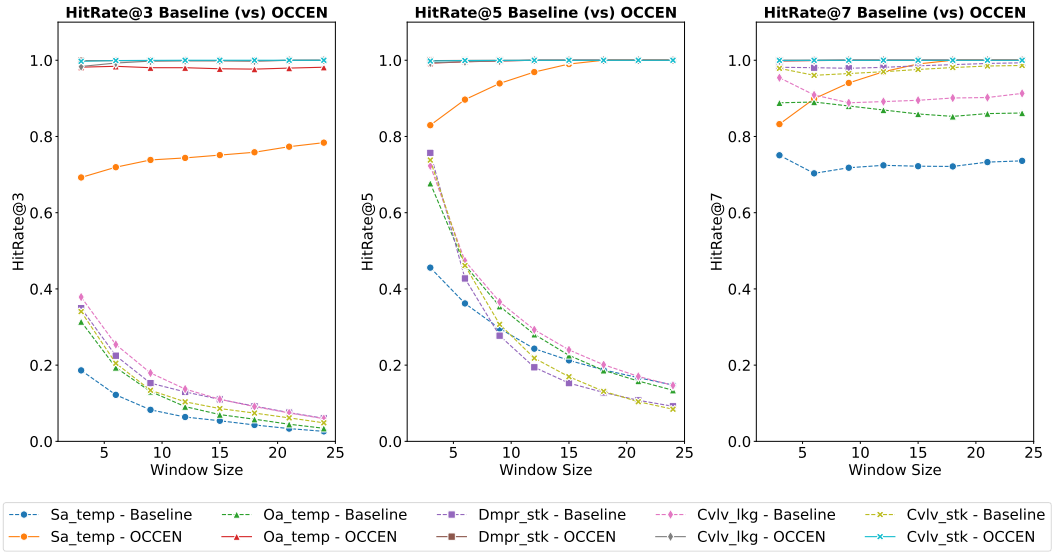


Figure 6 Mean score comparison of HitRate@k metric for baseline and OCCEN.

For the fault type *Cvlv_lkg*, OCCEN also demonstrates better performance compared to the baseline across all k-values and window sizes. The hit rates for the OCCEN are significantly higher, with perfect hit rates at k = 7 for window sizes of 12 and beyond. The baseline, however, achieves lower hit rates. Lastly, the fault type *Cvlv_stk* showcases a clear trend where OCCEN outperforms the baseline. The hit rates for OCCEN remain consistently high across all window sizes, reaching up to 100% for k = 5 and 7 across multiple window sizes. The baseline performance remains similar to the *Cvlv_lkg* performance. This is mainly because these points are part of the same component, i.e., cooling coil. Overall, the OCCEN consistently outperforms the baseline method in diagnosing the various fault types, as evidenced by higher *HitRate@k* metrics for all examined values of k and across different window sizes.

5.3.3 RootCause@k

Remember, in this study, each fault type has only one true cause. Therefore, in this evaluation procedure we are particularly interested in measuring how accurately the true fault cause is identified within the top-k ranked diagnoses. Figure 7 provides an overview of the outcomes and at first glance it can be observed that OCCEN performs better at ranking true diagnoses than the baseline.

Looking closely, for the fault type *Sa_temp*, the OCCEN method shows steady improvement and this gradually gets better as k increases. However, the baseline showing minimal improvement, if any, while OCCEN continues to perform well. Notably, OCCEN achieves a maximum RootCause@3 score of 98%, indicating significant improvement over the baseline. In contrast, the baseline method performs poorly across all window sizes and k values. Interestingly, as the window size increases, the baseline's performance declines further, with the RootCause@3 score dropping to 2% at a window size of 24. The performance disparity between the baseline and OCCEN is even more pronounced for the fault type *Oa_temp*. OCCEN begins with a score of 36% at a window size of 3 and gradually declines to 16% at a window size of 24. Despite this decline, OCCEN consistently outperforms the baseline across all window sizes and k values. OCCEN more consistently ranks the root cause for *Oa_temp* within the top 5 diagnoses compared to the baseline. In contrast, the baseline starts with a score of 1% at a window size of 3, which decreases as k increase. The baseline struggles to adapt as the window size increases, highlighting its limitations in ranking the root cause effectively.

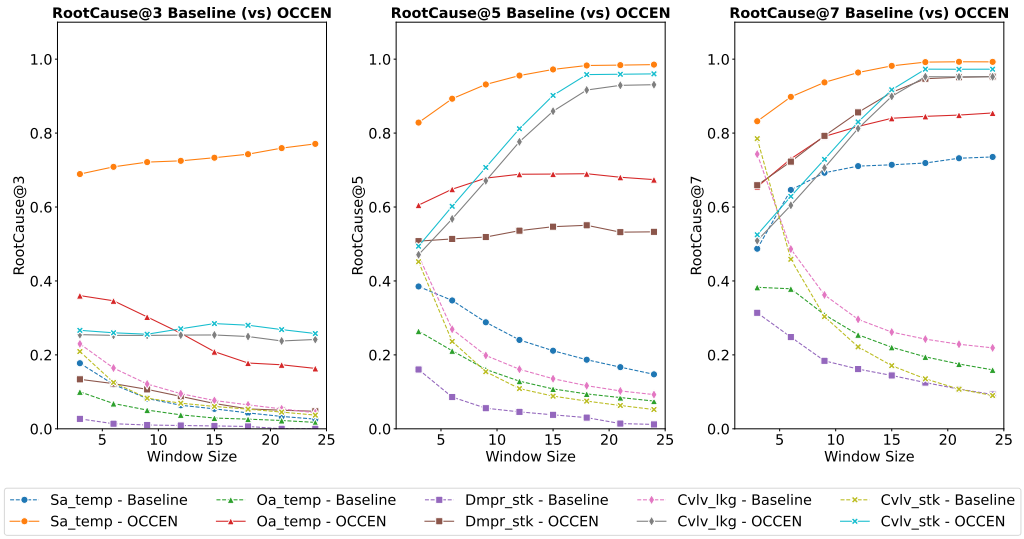


Figure 7 Mean score comparison of RootCause@k metric for baseline and OCCEN.

For the fault type *Dmpr_stk*, OCCEN also struggles to rank the root cause within the top 3 diagnoses. However, for the RootCause@5 metric, OCCEN improves significantly, with scores between 50% and 53%, while the baseline's performance declines. The baseline method performs poorly on the RootCause@3 metric across all window sizes, scoring as low as 2% at a window size of 3 and staying below 1% even as the window size increases. For the RootCause@7 metric, the baseline's performance remains weak, starting at 31% and dropping to 9%. In contrast, OCCEN shows strong performance, starting at 65% and reaching 95% demonstrating its ability to rank the true cause among the top seven diagnoses.

For the fault type *Cvlv_lkg*, the baseline starts relatively well with a RootCause@3 score of 22% but its performance declines to 4%. In contrast, OCCEN consistently improves, particularly for larger k values and window sizes. The overall trend of OCCEN significantly improves for larger k values in contrast to baseline. For RootCause@5, the baseline begins at 46% but drops to 9% as the window size increases, whereas OCCEN starts at 47% and rises to 93%. Similarly, for RootCause@7, the baseline initially scores 74% but falls to 21%, while OCCEN improves from 50% to 95%. The pattern is consistent for the fault type *Cvlv_stk*,

where OCCEN significantly outperforms the baseline for larger k values and window sizes, achieving a RootCause@5 score of 96% and a RootCause@7 score of 97% at a window size of 24.

In summary, OCCEN more frequently ranks the root cause at higher positions compared to the baseline, showcasing its diagnostic capability by learning cluster representations and assigning relevant features (diagnoses) to the system's faulty operational conditions. This analysis is also consistent with the overlapping diagnoses and OCCEN's ability to rank at least one true diagnosis at the top k places.

6 Implications & Threats to Validity

In this section, we describe the implications and limitations of our work.

6.1 Implications

In real-world applications, especially dynamic systems, labeled anomalous instances are frequently unavailable due to the substantial time and effort required for their collection. Consequently, our work focuses on leveraging only the normal operations of complex dynamic systems to design an effective anomaly detection and diagnosis framework. We achieve this by combining data-driven techniques in a novel manner. Additionally, this work prioritizes and addresses the requirements of the operators, who often seek a list of diagnoses to identify and pinpoint the root cause of issues within the system. Furthermore, the framework's design enables the integration of various data-driven learning methodologies, e.g., clustering, and classification, including advanced techniques like deep learning. However, for this study, we focus solely on traditional learning methods to understand the distribution of normal system operations and to learn cluster representations for associating features with faulty states.

6.2 Threats to Validity

Like any other research work, our work has its limitations. We evaluated our approach against a simple baseline to demonstrate its effectiveness. The rationale behind this evaluation method was to showcase its practical applicability for anomaly detection and diagnosis. To our knowledge, there has been limited work in the existing literature focused on this specific use case, particularly regarding diagnostic functionality that provides a ranked list of diagnoses. While we report our findings for a multivariate time series dataset related to an air handling unit, further evaluation on other real-world datasets is needed to assess generalizability. Further, determining an optimal number of clusters for real-world applications are necessary and planned for future work. Finally, the FastDTW step relies on a reference signal of "normal" behavior, which is often unavailable without a simulation model or historical normal data. Despite these limitations, we believe the proposed framework has strong potential for similar problems.

7 Conclusion

In this study, we developed and proposed an anomaly detection and diagnosis framework called OCCEN. OCCEN relies solely on non-anomalous instances of multivariate time series. Learning cluster representation of anomalies, where each cluster can represent a fault type, the framework enhances feature significance and associations within each cluster. A ranked list of diagnoses is produced for each cluster assignment (i.e., anomaly) by employing

the XAI method (LIME) and the sequence matching technique (FastDTW) to consider temporal dependencies. The extensive evaluation of OCCEN on a synthetic dataset of the real-world use case of an air handling unit demonstrated that OCCEN outperforms the classical baseline method.

References

1 Anam Abid, Muhammad Tahir Khan, and Javaid Iqbal. A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review*, 54:3639–3664, 2021. doi:10.1007/S10462-020-09934-2.

2 Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, pages 8–15, 2013.

3 Yeonjin Bae, Saptarshi Bhattacharya, Borui Cui, Seungjae Lee, Yanfei Li, Liang Zhang, Piljae Im, Veronica Adetola, Draguna Vrabie, Matt Leach, et al. Sensor impacts on building and hvac controls: A critical review for building energy performance. *Advances in Applied Energy*, 4:100068, 2021.

4 Anna M Bartkowiak. Anomaly, novelty, one-class classification: a comprehensive introduction. *International Journal of Computer Information Systems and Industrial Management Applications*, 3(1):61–71, 2011.

5 A Beghi, R Brignoli, Luca Cecchinato, Gabriele Menegazzo, Mirco Rampazzo, and F Simmini. Data-driven fault detection and diagnosis for hvac water chillers. *Control Engineering Practice*, 53:79–91, 2016.

6 Efrem Heri Budiarto, Adhistya Erna Permanasari, and Silmi Fauziati. Unsupervised anomaly detection using k-means, local outlier factor and one class svm. In *2019 5th international conference on science and technology (ICST)*, volume 1, pages 1–5. IEEE, 2019.

7 Xuewu Dai and Zhiwei Gao. From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis. *IEEE Trans. Ind. Informatics*, 9(4):2226–2238, 2013. doi:10.1109/TII.2013.2243743.

8 Laura Erhan, M Ndubuaku, Mario Di Mauro, Wei Song, Min Chen, Giancarlo Fortino, Ovidiu Bagdasar, and Antonio Liotta. Smart anomaly detection in sensor systems: A multi-perspective review. *Information Fusion*, 67:64–79, 2021. doi:10.1016/J.INFFUS.2020.10.001.

9 Astha Garg, Wenyu Zhang, Jules Samaran, Ramasamy Savitha, and Chuan-Sheng Foo. An evaluation of anomaly detection and diagnosis in multivariate time series. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2508–2517, 2021. doi:10.1109/TNNLS.2021.3105827.

10 Zhiqiang Ge. Review on data-driven modeling and monitoring for plant-wide industrial processes. *Chemometrics and Intelligent Laboratory Systems*, pages 16–25, 2017.

11 Jessica Granderson, Guanjing Lin, Yimin Chen, Armando Casillas, Jin Wen, Zhelun Chen, Piljae Im, Sen Huang, and Jiazhen Ling. A labeled dataset for building hvac systems operating in faulted and fault-free states. *Scientific Data*, 2023.

12 H Burak Gunay and Zixiao Shi. Cluster analysis-based anomaly detection in building automation systems. *Energy and Buildings*, 228:110445, 2020.

13 Fouzi Harrou, Abdelkader Dairi, Bilal Taghezouit, and Ying Sun. An unsupervised monitoring procedure for detecting anomalies in photovoltaic systems using a one-class support vector machine. *Solar Energy*, 179:48–58, 2019.

14 Andreas Holzinger, Anna Saranti, Christoph Molnar, Przemyslaw Biecek, and Wojciech Samek. Explainable ai methods-a brief overview. In *International workshop on extending explainable AI beyond deep models and classifiers*, pages 13–38. Springer, 2022.

15 Rolf Isermann. *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Springer Science & Business Media, 2011.

16 Neha Kant and Manish Mahajan. Time-series outlier detection using enhanced k-means in combination with pso algorithm. In *Engineering Vibration, Communication and Information Processing: ICoEVCI 2018, India*, pages 363–373. Springer, 2019.

17 Shehroz S Khan and Michael G Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, 2014. doi:10.1017/S026988891300043X.

18 Zhiling Lan, Ziming Zheng, and Yawei Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, 2009. doi:10.1109/TPDS.2009.52.

19 Jinbo Li, Hesam Izakian, Witold Pedrycz, and Iqbal Jamal. Clustering-based anomaly detection in multivariate time series data. *Applied Soft Computing*, 100:106919, 2021. doi:10.1016/J.ASOC.2020.106919.

20 Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008. doi:10.1109/ICDM.2008.17.

21 Hang Liu, Youyuan Wang, and WeiGen Chen. Anomaly detection for condition monitoring data using auxiliary feature vector and density-based clustering. *IET Generation, Transmission & Distribution*, 14(1):108–118, 2020.

22 Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

23 James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

24 Luigi Martirano and Massimo Mitolo. Building automation and control systems (bacs): a review. In *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*, pages 1–8, 2020.

25 Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. Autoencoders. In *Machine learning*, pages 193–208. Elsevier, 2020.

26 Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663), 2009.

27 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

28 Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007. URL: http://content.iospress.com/articles/intelligent-data-analysis/ida00303.

29 Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001. doi:10.1162/089976601750264965.

30 Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

31 Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1409–1416, 2019. doi:10.1609/AAAI.V33I01.33011409.

Property Learning-Based Fault Detection for Liquid Propellant Rocket Engine Control Systems

Andrea Urgolo[1]
Silicon Austria Labs GmbH (SAL), Graz, Austria

Ingo Pill
Institute of Software Technology, Graz University of Technology, Austria

Günther Waxenegger-Wilfing
University of Würzburg, Germany
German Aerospace Center (DLR), Lampoldshausen, Germany

Manuel Freiberger
Silicon Austria Labs GmbH (SAL), Graz, Austria

Abstract

Accommodating the dynamic and uncertain operational environments that are typical for aerospace applications, our work focuses on robust fault detection and accurate diagnosis in the context of Liquid Propellant Rocket Engines. To this end, we employ techniques based on learning temporal properties which are then dynamically adapted and refined based on observed behavior. Leveraging the capabilities of genetic programming, our methodology evolves and optimizes temporal properties that are validated through formal methods in order to ensure precise, interpretable real-time fault monitoring and diagnosis. Our integrated strategy enables us to enhance resilience, safety and reliability when operating rocket engines – due to the proactive detection and systematic analysis of operational deviations before they would escalate into critical failures. We demonstrate the effectiveness of our method via a rigorous evaluation across varied simulated fault conditions, in order to showcase its potential to significantly mitigate the fault-related risks in aerospace systems.

2012 ACM Subject Classification Computing methodologies → Machine learning; Computer systems organization → Embedded and cyber-physical systems

Keywords and phrases Machine learning, Runtime verification, Property learning, Monitoring, Fault detection, Diagnosis, Genetic programming, Explainable AI

Digital Object Identifier 10.4230/OASIcs.DX.2024.15

Funding This project has been financially supported by the Austrian Research Promotion Agency (FFG) under grant no. 897989.

1 Introduction

With the visionary goal to become the first climate-neutral continent by 2050 [17], the European Union claims a global leadership role in the world-wide sustainability efforts. In the European aerospace sector, we still see a strong need for new technology that allows us to contribute to implementing this ambitious agenda. This need naturally fosters the advent of reusable rocket technology, such as to minimize the ecological footprint and maximize resource utilization. SpaceX's Falcon 9 [32] exemplifies the feasibility of reusable launch vehicles even today and illustrates a tangible path towards a more sustainable space sector.

The deployment of reusable rocket engines requires precise and dynamic control algorithms for Liquid Propellant Rocket Engines (LPREs) that need to be capable of compensating large and systematic deviations from expected behavior. In particular, intelligent control

[1] Corresponding author


© Andrea Urgolo, Ingo Pill, Günther Waxenegger-Wilfing, and Manuel Freiberger;
licensed under Creative Commons License CC-BY 4.0



approaches must (a) choose the operating point considering component wear (life extension), (b) robustly address internal and external disturbances, and (c) enable a seamless transition between alternate operation modes – during different flight phases, but also to mitigate issues.

Such intelligent control stands in contrast to classic rocket control designs though. Those rely mostly on open-loop control or, at best, steady-state controllers (see [31] for an overview) – requiring us to evolve our current concepts. Achieving intelligent control in a resource-limited LPRE scenario naturally entails the integration of resource-optimal, quick, and resilient monitoring and diagnosis systems. These systems must provide critical data by tracking component wear, detecting anomalies, and identifying system faults that are paramount for a comprehensive understanding of the current state of the entire system. This awareness is essential for enabling effective closed-loop control with fault management capabilities.

In this paper, we focus on monitoring LPREs and investigate the applicability of learning monitorable properties for fault detection. To that end, we develop and evaluate a property learning-based approach for fault detection and isolation, leveraging temporal logic and genetic programming techniques. Our contributions are as follows:

- a multi-objective genetic programming approach for automatically learning fault detection/isolation properties in Signal Temporal Logic (STL),
- its application to monitoring and diagnosing LPREs, enabling the detection and isolation of various fault types, including sensor drifts, offsets, and mechanical failures,
- a method for exploiting virtual sensing in the fault detection/isolation process,
- a comprehensive evaluation of our technology.

We structured our presentation as follows: We start with an introduction of preliminaries in Sec. 2. After discussing our approach and contributions in Sec. 3, we introduce the LPRE control application (domain) in Sec. 4, in which we furthermore discuss our experiments and evaluate the effectiveness of our methodology. A detailed analysis of the strengths and weaknesses is provided in Sec. 5, followed by a discussion of related work in Sec. 6. Finally, we summarize our main findings as well as outline future research directions in Sec. 7.

2 Preliminaries

In this section, let us introduce essential concepts and our notation. Covered topics include the Signal Temporal Logic and the basics of monitoring as well as evolutionary algorithms.

2.1 The Signal Temporal Logic STL

Similar to the well-known Linear Temporal Logic (LTL) [30], Signal Temporal Logic (STL) [23] enhances propositional logic with temporal modalities that allow us to reason about the future. Unlike LTL, which requires the discretization of signals and time, STL allows direct reference to continuous-valued signals and continuous real-time in formulae, making it particularly interesting for real-time monitoring and analyzing signals from cyber-physical systems (CPS). By leveraging STL, we can define and monitor nuanced temporal properties that are critical for assessing a system's behavior, ensuring that any deviation from the expected performance is detected promptly. This is essential for maintaining the safety and reliability of rocket engines in dynamic conditions, where even minor anomalies can have significant consequences.

Before defining a trace as aggregation of observed signals, we briefly introduce parts of our notation. Let $\top$ and $\bot$ denote the constants *True* and *False*, and let $\neg$ denote negation. A signal τ defines for each point in time $0 \leq t \leq T(\tau)$ a real value, where the temporal

horizon $T(\tau)$ of signal τ may be finite (so that the execution ends at $T(\tau)$), or infinite. For continuous time, some signal τ can thus be described by a function $f_\tau : \mathbb{R}_0^+ \to \mathbb{R}$ that maps a (non-negative) point in time to some (real) signal value. While STL can operate in a continuous domain, practical implementations often discretize time or restrict the signal values to rational numbers for computational reasons.

In our work, we will consider (clocked) time series data with a finite temporal horizon – which we will also refer to as a *finite trace*. Now, let us furthermore assume that we observe for each point in time all the signal values in a single vector π, such that we refer with (a) Π to the entire trace (so to say the sequence of vectors for the individual time steps), (b) $\pi = \Pi(t)$ to the signal vector at time t, (c) Π_i to the trace of signal i, (d) $\Pi_i(t)$ to the value of signal i at time step t, and (e) $\Pi[j, k]$ to a partial trace that is restricted to the time interval $[j, k]$. For a finite trace, we can thus define signals as finite sequences of signal values (one per $T(\Pi) + 1$ time steps), or as a function mapping each element in the sequence $[0, \ldots, T(\tau)]$ to a value in $\mathbb{R}$. Hereafter, for convenience, the length of a signal τ will be denoted by $len(\tau)$, equal to $T(\tau) + 1$. For an infinite temporal horizon and clocked time, we can define signals and traces either as functions mapping from $\mathbb{N}$ to $\mathbb{R}$ or $\mathbb{R}^{|\tau|}$ respectively, or as finite sequences with a finite stem and an infinite loop (k,l loops) like in model-checking [12]. With this notation, let us now define the syntax and semantics of STL:

▶ **Definition 1.** *Let $\mu = f(\Pi(t)) > c$ be an atomic predicate that compares a function over the signal values for some time step t to a real-valued constant c. With φ and ψ being* STL *formulae, and I being one of the intervals (a, b), $(a, b]$, $[a, b)$, or $[a, b]$ for two points in time such that $a \leq b$, we can define the syntax of* STL *inductively as*

$$\varphi ::= \top \mid \mu \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \,\mathsf{U}_I\, \psi$$

▶ **Definition 2.** *Let I_t be the interval of an until operator applied at time step t. The satisfaction of an* STL *formula φ as of Def. 1 by a trace Π at time t is then defined as follows*

$$\begin{aligned}
&\Pi(t) \models \mu && \text{iff } f(\Pi(t)) \geq c \\
&\Pi(t) \models \neg\varphi && \text{iff } \neg(\Pi(t) \models \varphi) \\
&\Pi(t) \models \varphi \wedge \psi && \text{iff } \Pi(t) \models \varphi \wedge \Pi(t) \models \psi \\
&\Pi(t) \models \varphi \,\mathsf{U}_I\, \psi && \text{iff } \exists t' \in I_t \text{ s.t. } \Pi(t') \models \psi \wedge \forall t'' \in [t, t'] : \Pi(t'') \models \varphi
\end{aligned}$$

As with LTL, we can introduce syntactic sugar that does not increase STL's expressiveness but improves its usability and intuitiveness. Prominent examples include the *or* $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, *exclusive or* $\varphi \oplus \psi = \varphi \wedge \neg\psi \vee \neg\varphi \wedge \psi$, and *implication* $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ on the logic level, and temporal operators like *finally* a.k.a. *eventually* for denoting $F_I\, \varphi = \top \,\mathsf{U}_I\, \varphi$, or *globally* a.k.a. *always* for denoting $G_I\, \varphi = \neg(F_I\, \neg\varphi)$.

Let us stress that for our definitions we adopt a simplified view of having only a scalar value per signal. It is easy to see that we can easily accommodate also multiple values for one signal (this does not change τ), or modify the definitions such that signals can take values other than reals numbers (affecting only μ); e.g., a 64-bit data signal interpreted at the logic or voltage level, can be split into individual signals for each data line. Please note that Def. 2 offers very simple STL semantics, with alternative ones discussed in the following subsection.

2.1.1 Monitoring STL: Bounded STL Formulae

Traditional offline verification methods, such as model checking or testing, enable comprehensive analysis of a system's behavior. Model checking, for instance, allows us to prove or falsify properties expressed in some temporal logic for a system's entire behavior. Testing involves

stimulating a system with specific inputs (defined in test cases) and observing the resulting behavior. In either case, we investigate complete executions or runs, i.e., the observable part coined as *trace*. However, when monitoring a system at run-time, we only have a finite prefix of an execution available. In contrast to implementations of a test oracle [29], we thus have to implicitly or explicitly consider the entire set of executions that extend this prefix.

Consequently, monitoring not only entails different semantics of expressed properties [5, 29], but also presents the challenge that not all properties are indeed monitorable. A property is *positively monitorable* if any satisfying trace can *conclusively* demonstrate this through a finite prefix, while it is *negatively monitorable* if any violating trace provides such evidence within a finite prefix. Safety properties, for instance (see [3] for a discussion of safety and liveness), are negatively monitorable because any trace showing a bad event violates the property, meaning that such events should never occur. Conversely, co-safety properties are positively monitorable since a finite trace can affirm a positive event within its scope.

As for LTL [5], there are alternative semantics that are of special interest for monitoring purposes, like the following quantitative one taken from [14]. The idea is that atomic predicates do not evaluate to $\top$ or $\bot$, but to a real value, i.e., $f(\Pi(t)) - c$ from μ in Def. 1, propagated along the entire formula. These quantitative semantics provide a robustness measure ρ, which indicates the degree to which a signal satisfies a formula at a given time.

▶ **Definition 3.** *The robustness ρ of an* STL *formula φ for a trace Π at time t is defined as:*

$$
\begin{aligned}
\rho(\top, \Pi, t) &= +\infty \\
\rho(\Pi_i \geq c, \Pi, t) &= \Pi_i(t) - c \\
\rho(\neg\varphi, \Pi, t) &= -\rho(\varphi, \Pi, t) \\
\rho(\varphi_1 \wedge \varphi_2, \Pi, t) &= \min\{\rho(\varphi_1, \Pi, t), \rho(\varphi_2, \Pi, t)\} \\
\rho(\varphi_1 \mathsf{U}_I \varphi_2, \Pi, t) &= \max_{t_1 \in t+I} \min\{\rho(\varphi_2, \Pi, t_1), \min_{t_2 \in [t, t_1)} \rho(\varphi_1, \Pi, t_2)\}
\end{aligned}
$$

With the latter concept outlined, we can now formalize the definition of STL monitoring.

▶ **Definition 4.** *Let $\mathcal{X}$ (resp., $\bar{\mathcal{X}}$) be the set of infinite signals (resp., finite signals), and $\mathcal{C}(\Pi)$, the set of completions of a given finite signal $\Pi \in \bar{\mathcal{X}}$ over $\{0, \ldots, t\} \subset \mathbb{N}$, defined as $\mathcal{C}(\Pi) = \{\hat{\Pi} \in \mathcal{X} \mid \hat{\Pi}(t') = \Pi(t')$ for all $t' \in \mathbb{N}, 0 \leq t' \leq t\}$.* STL *monitoring is defined by the function $mon : \bar{\mathcal{X}} \times \mathsf{STL} \to \top, \bot, ?$ s.t. $mon(\Pi, \varphi)$ returns $\top$ iff $\hat{\Pi}, 0 \models \varphi$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, $\bot$ iff $\hat{\Pi}, 0 \not\models \varphi$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, equivalent to $\rho(\varphi, \hat{\Pi}, 0) < 0$ for all $\hat{\Pi} \in \mathcal{C}(\Pi)$, or ? otherwise.*

Currently, there are no tools that support the monitoring of unbounded STL formulas as defined in our work. However, bounded-time STL (bSTL), a specific fragment of STL, supports monitoring in practical applications [26]. This is achieved by restricting the temporal modality time interval I in bSTL to finite boundaries, specifically to $I = [a, b]$, where $a, b \in \mathbb{N}$ are both finite. bSTL formulas are characterized by a temporal *horizon* $H(\phi)$, which quantifies the maximum future time span necessary to evaluate the truth of the formula. For instance, the formula $\phi = x \geq 3\ \mathsf{U}_{[0,3]}\ x \geq 5$ has a horizon of 3, indicating that its validation requires data spanning at least three time units from the initial point of evaluation.

▶ **Definition 5.** *Formally, the* bSTL *formula's temporal horizon $H : \mathsf{bSTL} \to \mathbb{N}$ is defined as: (i) $H(\top) = 0$; (ii) $H(\Pi_i \geq c) = 0$; (iii) $H(\neg\varphi) = H(\phi)$; (iv) $H(\varphi_1 \wedge \varphi_2) = \max\{H(\varphi_1), H(\varphi_2)\}$; (v) $H(\varphi_1\ \mathsf{U}_{[a,b]}\ \varphi_2) = b + \max\{H(\varphi_1) - 1, H(\varphi_2)\}$.*

Furthermore, to address real-world monitoring scenarios, which require responsive system behavior analysis where conditions may need to be evaluated at any given point across a signal, we extend monitoring capabilities to apply to any suffix beyond the horizon.

▶ **Definition 6.** bSTL *monitoring operates through the function eb-mon* : $\bar{\mathcal{X}} \times \mathsf{bSTL} \to \{\top, \bot, ?\}$*, which relies on the function mon from Def. 4, and is defined as:*

$$\textit{eb-mon}(\Pi, \varphi) = \begin{cases} ? & \textit{if } len(\Pi) < H(\varphi) + 1 \\ \bigvee_{0 \le i \le len(\Pi)-1-H(\varphi)} mon(\Pi[i, len(\Pi)-1], \varphi) & \textit{otherwise} \end{cases}$$

This formulation ensures that a monitor can definitively issue truth values $\top$ or $\bot$, or an undefined verdict ? if the horizon has not yet been reached, thus safeguarding against premature evaluations. The effectiveness of this extended monitoring is captured by the tool `rtamt` [26], which supports practical implementation of bSTL monitoring for both discrete and dense-time properties, facilitating robust and reliable system evaluations.

Fault detection in system monitoring typically involves identifying anomalies or operational deviations that could lead to system failures. In the following, the concept of fault detection grounded on the definitions provided in the existing literature on system verifiability and monitorability [1, 21], is formalized as it is applied within the scope of our study.

▶ **Definition 7.** *Given a set $\mathcal{P}$ of properties formulated in* bSTL*, a fault is detected on an execution trace $\Pi \in \bar{\mathcal{X}}$ at a time step $i \in \mathbb{N}$ if there exists at least one fault property $\psi \in \mathcal{P}$ such that the monitoring function eb-mon$(\Pi[0, i], \psi)$ returns $\top$. Here, $\bar{\mathcal{X}}$ represents the set of all considered (finite) engine run traces, and $\Pi[0, i]$ denotes the prefix of the trace Π from the start up to the i-th time step.*

This formalization captures the essence of real-time fault detection by leveraging the temporal properties specified in bSTL, allowing for the proactive identification and subsequent handling of system anomalies before they escalate into more severe problems.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are population-based metaheuristics inspired by natural evolution, effectively addressing combinatorial optimization challenges by using historical data to guide search efforts towards promising areas [16]. These algorithms involve a population of individuals, each representing a solution to an optimization problem, evaluated through a fitness function based on adaptation criteria. This fitness may be assessed via single- or multi-objective approaches, depending on the problem's complexity. The evolutionary cycle advances through generations, with a selection mechanism favoring higher fitness individuals for reproduction. Genetic operations such as crossover and mutation facilitate diversity and prevent premature convergence by introducing variations. The process iterates until meeting predefined criteria such as optimal fitness or maximum generation count.

This research specifically utilizes genetic programming (GP), a branch of EAs where individuals are represented by means of computational trees evolving towards optimal solutions over successive generations through tree-specific genetic operations. Additionally, this study leverages multi-objective EAs to optimize multiple objectives simultaneously, generating diverse Pareto-optimal solutions. This approach enhances solution exploration, allows customization of the evolutionary process, supporting a detailed investigation into optimal configurations for the extraction of complex properties in diverse scenarios. The algorithm used in this study, as previously defined in [7], is implemented through DEAP (Distributed Evolutionary Algorithms in Python), a versatile framework for rapid prototyping of EAs [18]. Operationally, two sets containing nominal and anomalous finite traces are processed, and a bSTL formula is subsequently derived to distinguish between these two conditions. Below, we present the main implementation characteristics of this algorithm.

Population and its Initialization. The population within our evolutionary algorithm framework consists of individuals, each represented as a computation tree φ encoding a syntactically correct bSTL formula. The population is generated using the *genHalfAndHalf* method from DEAP, which adheres to guidelines established by Koza [20]. This method ensures diversity in the tree structures by alternating between generating trees with uniform leaf depths and trees where leaves may vary in depth.

Nodes of the Computation Tree. In the evolutionary framework, each node within a computation tree can represent various structural and functional components of a bSTL formula. Nodes primarily encapsulate constraints such as $\Pi_i \geq c$, where Π_i is a signal identifier and c is a constant. They can also represent complex bSTL formulas where the outermost operator is temporal, for example, $\varphi \mathsf{U}_{[a,b]} \psi$, or Boolean expressions like $\varphi \vee \psi$. Here, φ and ψ themselves are bSTL sub-formulas, recursively structured as trees within the main computation tree. Moreover, nodes may embody terminal values crucial for the construction and evaluation of formulas such as: (*i*) interval bounds of temporal operators, expressed as $I = [a, b]$, where $a, b \in \mathbb{N}$ and $a \leq b$; (*ii*) signal identifiers Π_i, with $1 \leq i \leq |\Pi|$, linking the formula directly to the specific components of a multi-dimensional signal; (*iii*) constants c, used within constraints, where each c is chosen from the domain $Dom(\Pi_i)$ of the signal component Π_i. To facilitate the dynamic and ephemeral representation of values such as constants and interval bounds, DEAP employs *EphemeralConstants*. These are particularly useful for encoding values that are not fixed but need to be generated anew for each instantiation of the tree, thus supporting a rich diversity in the evolutionary process.

Crossover. The crossover operation is a key genetic mechanism in evolutionary algorithms, used to generate new individuals from existing parent solutions. In our implementation, this process involves a one-point crossover technique facilitated by DEAP's *cxOnePoint* function. This function selects a random node within the computation tree of each parent and exchanges the subtrees rooted at these nodes, thereby producing two offspring with combined genetic traits from both parents. To manage the complexity of the resulting computation trees and prevent bloat – a growth in program size without fitness improvement – a static height limit of 17 is imposed on the offspring's computation trees (DEAP's *staticLimit*), once again following Koza's guideline [20]. If this limit is exceeded, the child is discarded and replaced by a randomly selected parent. Similarly, the crossover process might occasionally produce invalid individuals, especially in terms of computation tree's horizon. Specifically, if an offspring's formula φ results in a horizon $H(\varphi)$ that exceeds the permissible range (i.e., $H(\varphi) \geq len(\Pi)$), this offspring is deemed non-viable and replaced by a randomly chosen parent. This strategy ensures that only viable solutions with valid and effective genetic configurations persist in the population for further evolutionary processing.

Mutation. Mutation in evolutionary algorithms introduces variability and helps explore new genetic landscapes. In our implementation, two mutation operators are employed, each selected with equal probability to modify individuals within the population. The first operator, *mutNodeReplacement*, targets a randomly chosen node in the computation tree of an individual and replaces it with another node that maintains the tree's syntactical correctness. The second operator, *mutEphemeral*, is designed to alter the value of a single constant in the individual's tree. Consistent with our approach to manage program complexity and avoid bloat, the same *staticLimit* is applied to the height of the trees after mutation, mirroring the constraints imposed during the crossover process. Post-mutation, it is crucial to verify

the validity of the resulting individual. If the individual's horizon or height does not align properly with the problem's constraints, the mutation is considered unsuccessful, and the original pre-mutation individual is retained in the population, hence preserving the overall quality and relevance of the evolving population.

Selection. To foster diversity within the population, our approach utilizes the elitist selection strategy as outlined in NSGA-III [13]. This method incorporates *reference points* and *niche preservation* principles, ensuring a diverse set of solutions by adequately covering all areas of the Pareto front. For a detailed explanation of these concepts, readers are directed to the foundational work by Deb et al. [13]. Finally, before mutation and crossover, elite individuals from the previous generation are selected to form the elite set. This set is then added to the next population, ensuring the persistence of high-performing individuals and their contribution across the evolutionary process.

Termination Criteria and Extraction of Final Solutions. The termination criteria for our evolutionary algorithm are twofold. First, an upper limit is set on the number of generations, a standard practice in evolutionary computations to ensure the process concludes in a timely manner. Second, we employ an *early stopping* strategy based on the *hypervolume* indicator. This measure quantifies the volume covered by the Pareto front relative to a reference point and gauges the diversity and quality of the solutions [8]. Termination occurs if there is no improvement in the hypervolume over a predetermined number of generations. Once the evolutionary process is completed, the final solutions are extracted by analyzing the Pareto front. Initially, we filter this front to retain only those individuals whose formula φ achieves an accuracy over 0.5, thereby outperforming a random classifier. Among these viable solutions, the individual contributing most significantly to the hypervolume – indicative of its excellence in diversity and fitness – is selected for output.

Algorithm Hyperparameters. The configuration of hyperparameters for the evolutionary algorithm (EA) utilized in this study was methodically determined through a grid search approach, conducted on a synthetic dataset of binary labeled bSTL traces. These traces were distinctively marked by a diverse assortment of formulas, representing two separate classes. The optimal hyperparameters were selected as follows: *population size* was set at 100, after evaluating a range from 50 to 1000; *crossover probability* was established at 0.7, with tested values ranging from 0.5 to 0.8; *mutation probability* was dynamically set to $0.5/\sqrt[2]{num_gen}$, encouraging robust initial exploration of the search space, which tapers off to enhance exploitation of promising solutions as generations progress; for this hyperparameter values between 0.3 and 0.6 were tested; *maximum generations* was conservatively capped at 500 to ensure computational feasibility while allowing ample evolution within the population; *hypervolume early stopping* was activated after 25 generations without improvement, with the threshold tested at intervals from 10 to 50 generations; *elite rate* was set at 0.05, determining the size of the elite set as 0.05 times the *population size*. This parameter governs the proportion of the population considered elite, influencing the selection process by favoring individuals associated to a higher hypervolume for reproduction. Additionally, a critical hyperparameter specific to this EA implementation, which leverages bSTL and `rtamt`, is the *max horizon*. This parameter intuitively sets an upper boundary h on the horizon of the formulas explored by the EA, leading to the following implications: formulas are restricted to capturing phenomena that extend no more than $h+1$ time points; experimental observations have shown that the computational demand of `rtamt` increases more than linearly with

the horizon length of a formula. For the purposes of expediting experimentation, which includes multiple runs to gather statistically significant results, the *max horizon* was set to 15. Although this may appear restrictive, it remains adequate for extracting significant and effective properties from the data, while also adhering to latency requirements typically considered for control systems in LPREs. In more general applications, the appropriate *max horizon* should be determined by domain experts, considering the aforementioned factors.

3 Material and Methods

Let us now describe our genetic programming-based methodology for learning STL properties that are capable of distinguishing between nominal and anomalous conditions, as well as the salient characteristics of the simulated dataset employed in our evaluation.

3.1 Our STL Property Learner

As outlined in the introduction, we use a genetic programming-based property learner, building on previous work discussed in Section 2.2. We extend this approach by employing a multi-objective fitness function that we define as follows.

To evaluate an individual in the population, the set of STL traces $\mathcal{X}$ is first "partitioned" into *nominal* ($\mathcal{X}_{neg}$) and *anomalous* ($\mathcal{X}_{pos}$) traces. This partitioning is necessary to distinguish between normal and faulty behavior, enabling the algorithm to effectively learn and optimize properties that are sensitive to anomalies while minimizing false positives from nominal traces. A multi-objective fitness function is then defined by using the `rtamt` monitoring algorithm for bSTL. Formally, our first objective measures how good a formula φ is in discriminating between nominal and faulty behavior traces.

▶ **Definition 8.** *For each trace* $\Pi \in \mathcal{X}$ *and each formula* φ*, the numerical counterpart of eb-mon*(Π, φ) *is the function* $NUM : \{\top, \bot, ?\} \to \{0, 1\}$ *s.t.* $NUM(\textit{eb-mon}(\Pi, \varphi)) = 1$ *if eb-mon*$(\Pi, \varphi) = \top$*, or* 0 *otherwise.*

▶ **Definition 9.** *The first objective measure, referred to as accuracy, is defined as follows:*

$$Acc(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{neg}} 1 - NUM(\textit{eb-mon}(\Pi, \varphi)) + \sum_{\Pi \in \mathcal{X}_{pos}} NUM(\textit{eb-mon}(\Pi, \varphi))}{|\mathcal{X}_{neg}| + |\mathcal{X}_{pos}|}$$

It is worth noting that, in order to maximize $Acc(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi)$, a formula φ should evaluate to $\bot$ on the nominal behavior traces and to $\top$ on the anomalous ones.

The second objective measures the robustness of the formula (normalized in the $[0, 1]$ interval) by means of bSTL quantitative semantics. As a preliminary step, at the beginning of the execution of the genetic algorithm, every signal in $\mathcal{X}$ is normalized in the $[0, 1]$ interval so that ρ ranges between -1 and 1. This step is handled implicitly and it does not alter the constant value c of constraints $\Pi_i \geq c$ in the generated output formula, which are still represented with their raw, non-normalized value.

▶ **Definition 10.** *This second objective, the robustness, is defined as follows:*

$$Rob(\mathcal{X}_{neg}, \mathcal{X}_{pos}, \varphi) = \frac{\begin{array}{r} |\mathcal{X}_{neg}| + |\mathcal{X}_{pos}| - \sum_{\Pi \in \mathcal{X}_{neg}} \max_{0 \leq i \leq len(\Pi) - 1 - H(\varphi)} \{\rho(\varphi, \Pi, i)\} \\ + \sum_{\Pi \in \mathcal{X}_{pos}} \min_{0 \leq i \leq len(\Pi) - 1 - H(\varphi)} \{\rho(\varphi, \Pi, i)\} \end{array}}{2 \cdot (|\mathcal{X}_{neg}| + |\mathcal{X}_{pos}|)}$$

The third objective measures the recall of the formula, denoted as *Rec*, which quantifies the ability to correctly identify all anomalous traces.

▶ **Definition 11.** *The recall is calculated as follows:*

$$Rec(\mathcal{X}_{pos}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{pos}} NUM(eb\text{-}mon(\Pi, \varphi))}{|\mathcal{X}_{pos}|}$$

In order to optimize $Rec(\mathcal{X}_{pos}, \varphi)$, the formula φ should evaluate to $\top$ for as many traces in $\mathcal{X}_{pos}$ as possible, effectively maximizing the detection of anomalous behavior.

The fourth objective measure, the specificity, evaluates the ability of a formula φ to accurately reject anomalous behavior in nominal traces.

▶ **Definition 12.** *The specificity is defined as follows:*

$$Spec(\mathcal{X}_{neg}, \varphi) = \frac{\sum_{\Pi \in \mathcal{X}_{neg}} 1 - NUM(eb\text{-}mon(\Pi, \varphi))}{|\mathcal{X}_{neg}|}$$

This measure is crucial for ensuring that φ maintains a high rejection rate of anomalous conditions in traces classified as nominal, thereby minimizing false positives. The ideal outcome for $Spec(\mathcal{X}_{neg}, \varphi)$ is a value close to 1, indicating that most nominal traces are correctly identified as such by φ.

The fifth objective measure, the parsimony, emphasizes the simplicity of the derived formulas, aiming to minimize their complexity.

▶ **Definition 13.** *The parsimony measure is defined as follows:*

$$Pars(\varphi) = 1 - \frac{height(\varphi)}{max\ height}$$

In this context, the function $height(\varphi)$ takes a bSTL formula φ and returns a natural number in $\mathbb{N}_+$ representing the height of the formula's computational tree. The term *max height* refers to a predefined static limit on the height of the computational trees within our genetic programming algorithm. This objective aims to minimize the complexity of the formulas by favoring those with shorter computational trees, thereby enhancing the understandability and efficiency of the derived models.

The sixth objective is the responsiveness which evaluates the promptness of a formula φ in reacting to changes in behaviors. Responsiveness is crucial for ensuring that the formula not only discriminates and measures robustly but also reacts in a timely manner to deviations from nominal behavior.

▶ **Definition 14.** *The responsiveness is formally defined as follows:*

$$Resp(\varphi) = 1 - \frac{H(\varphi)}{max\ horizon}$$

Here, $H(\varphi)$ represents the horizon of the formula φ, and *max horizon* the maximum horizon considered as defined in Def. 5. This metric aims to maximize the timeliness of the response by minimizing the horizon length relative to the maximal allowed horizon, thereby ensuring that φ is both effective and efficient in signaling anomalies.

To conclude, the seventh objective, the detection timeliness measure, quantifies the relative delay in detecting the farthest anomaly within the anomalous traces in $\mathcal{X}_{pos}$. This measure is critical in assessing the timeliness of φ in identifying anomalies, particularly in systems where detection delays may compromise functionality or safety.

▶ **Definition 15.** *Formally, the detection timeliness is defined as follows:*

$$Time(\mathcal{X}_{pos}, \varphi) = 1 - \frac{\max_{\Pi \in \mathcal{X}_{pos}} \{dect_point(\Pi, \varphi)\}}{\max_{\Pi \in \mathcal{X}_{pos}} \{len(\Pi)\}}$$

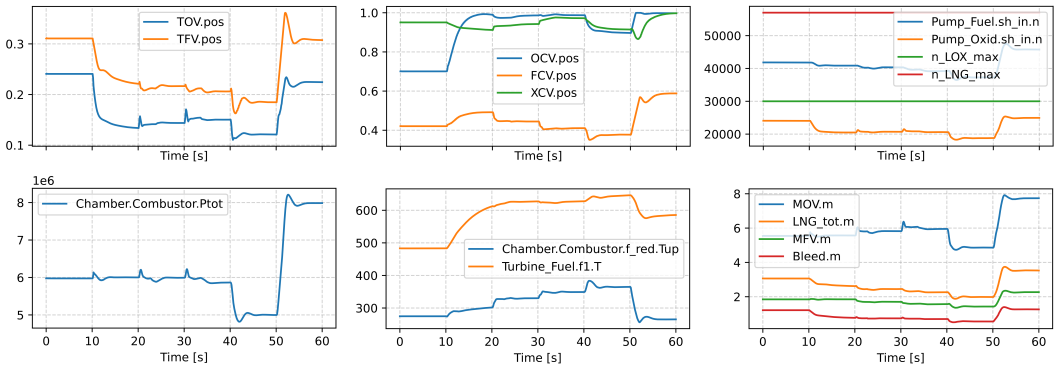


Figure 1 Simulated nominal run values for the engine under standard starting conditions.

In this definition, the detection point function, $dect_point(\Pi, \varphi)$, determines the earliest time point within a trace Π at which the formula φ is satisfied.

▶ **Definition 16.** *The detection point is calculated using the robustness measure ρ as follows:*

$$dect_point(\Pi, \varphi) = \min_{0 \leq i < len(\Pi)} (\{i \geq 0 \mid \rho(\varphi, \Pi, i) \geq 0\} \cup \{len(\Pi)\})$$

To maximize $Time(\mathcal{X}_{pos}, \varphi)$, φ should ideally detect anomalies as close to the start of each trace in $\mathcal{X}_{pos}$ as possible, thus ensuring minimal detection latency.

Since seven objectives are taken into consideration, no single best-performing solution can be directly selected from a given population by means of the fitness function. Rather, a *Pareto front* of optimal solutions can be identified, containing all *non-dominated* solutions.[2] Hence, once the evolutionary process is completed, the final solution, maximizing the hypervolume, is extracted from the Pareto front as previously described in Section 2.2.

3.2 Dataset

The dataset used in this work consists of transient state simulation of a representative pump-fed rocket engine of expander-bleed type similar to the LUMEN engine developed and tested by DLR Lampoldshausen [15, 33]. Each simulation was run for a total length of 60 s at a sampling frequency of 10 Hz. For a subset of runs, after a time of 15 s a fault had been injected. The faults considered were: a sensor drift, a sensor offset, a frozen sensor and a leakage in the methane sub-system. To formally define the dataset utilized in our study, let $\mathcal{X}$ represent the complete set of traces, where $\mathcal{X} = \mathcal{N} \cup \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$. Here, $\mathcal{N}$ denotes the set of traces simulated under nominal conditions, and $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$ represent the traces corresponding to the sensor drifting, sensor offset, sensor frozen, and leakage faults, respectively. Each subset $\mathcal{A}_i$ captures the distinct characteristics of the respective fault type, introduced to assess its impact on the system's performance and detectability.

The dataset captures various signals from the engine, categorized into several groups based on the type of measurement: *pressure sensors* measure the pressure inside the combustion chamber, at the inlet of the fuel and oxidizer turbines, and at the outlets from both the fuel

[2] A set $\mathcal{S}$ of solutions for an n-objective problem with fitness function $f = \langle f_i, \ldots, f_n \rangle$ is said to be *non-dominated* if and only if for each $x \in \mathcal{S}$, there exists no $y \in \mathcal{S}$ such that (i) $f_i(y)$ improves $f_i(x)$ for some i, with $1 \leq i \leq n$, and (ii) for all j, with $1 \leq j \leq n$ and $j \neq i$, $f_j(x)$ does not improve $f_j(y)$.

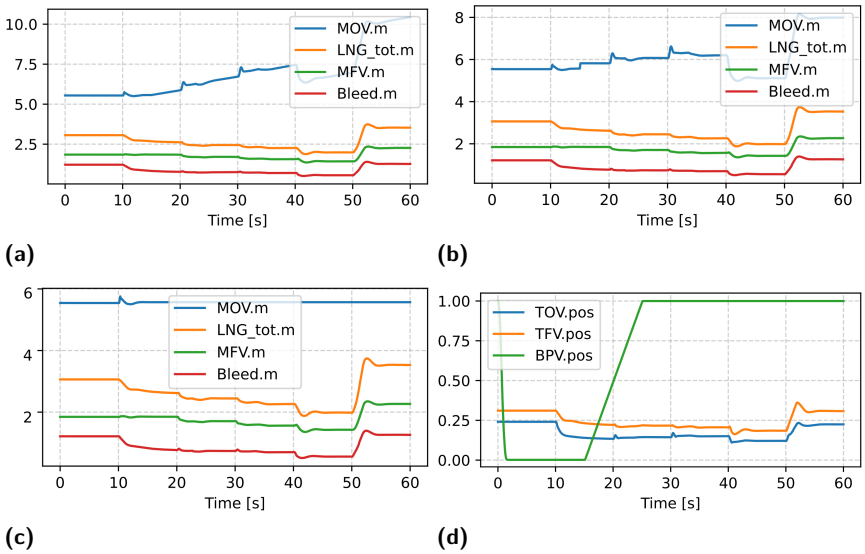


Figure 2 Simulated run values for the engine under standard starting conditions with the injection of faults: **(a)** sensor drift on `MOV.m`, **(b)** sensor offset on `MOV.m`, **(c)** sensor freezing on `MOV.m`, and **(d)** leakage on methane side (`BPV.pos`). For each case, the fault is injected after 15 seconds.

and oxidizer pumps, including inlet and outlet pressures of the cooling system; *temperature readings* are taken from the injected oxidizer and fuel and at the inlet and outlet of the cooling system, as well as at the input of both turbines; *valve positions* are recorded for the oxidizer control valve, fuel control valve, and the valves upstream of both turbines; finally, the *rotational speeds* of both the fuel and oxidizer pumps are also monitored.

Specifically, the features include `Chamber.Combustor.Ptot` representing the total pressure in the combustion chamber, `Chamber.Combustor.f_red.Tup` for the fuel injection temperature, `FCV.pos` indicating the fuel control valve position, `Chamber_Inj_red_dP_loss` and `Chamber_Inj_oxy_dP_loss` for pressure losses at fuel and oxidizer injection respectively. `Pump_Fuel.sh_in.n` and `Pump_Oxid.sh_in.n` measure the shaft velocity of the fuel and oxidizer turbopumps. `TFV.pos` and `TOV.pos` reflect the positions of the turbine fuel and oxidizer valves. Pressures at the inlets of fuel and oxidizer turbines are captured by `Turbine_Fuel.f1.P` and `Turbine_Oxid.f1.P`. `XCV.pos` logs the crossfeed control valve position, while `LOX_Inj.Pressure` and `Fuel_Inj.Pressure` represent the pressures at the liquid oxygen and fuel injection. `BPV.pos` denotes the position of the bypass valve, `MOV.m` measures the mass flow of the main oxidizer valve, `OCV.pos` indicates the oxidizer control valve position, `LNG_tot.m` represents the total mass flow of liquefied natural gas (LNG), `MFV.m` measures the flow rate through the main fuel valve, and `Bleed.m` captures captures the LNG mass flow that is not used for combustion.

These features were selected and analyzed to model the behavior of the propulsion system under nominal conditions and to identify deviations indicative of potential faults. An example of simulated values of these features for a nominal run of the engine under standard starting conditions is depicted in Figure 1, while Figure 2 illustrates four instances of fault injections performed on the `MOV.m` and `BPV.pos` features.

To rigorously evaluate the proposed methodologies for fault detection, the dataset $\mathcal{X}$ was randomly partitioned into training and testing splits to ensure unbiased assessment and validation of the learned properties. This split was done at the level of execution runs, allocating 80% of the data for training and 20% for testing. Formally, the dataset $\mathcal{X}$ was

divided into two subsets: $|\mathcal{X}_{\text{train}}| = 128$, with $|\mathcal{N}| = 64$, $|\mathcal{A}_1| = |\mathcal{A}_2| = |\mathcal{A}_3| = |\mathcal{A}_4| = 16$, and $|\mathcal{X}_{\text{test}}| = 32$, with $|\mathcal{N}| = 16$, $|\mathcal{A}_1| = |\mathcal{A}_2| = |\mathcal{A}_3| = |\mathcal{A}_4| = 4$. This structured approach to dataset division ensures a balanced representation of both nominal and anomalous conditions across the training and testing phases, providing a robust foundation for the evaluation of the fault detection algorithms developed in this study.

4 Experimentation

Our evaluation of the genetic algorithm's capabilities in distinguishing between nominal and faulty operational states under various predefined scenarios showcases our approach's applicability in real-world applications, where the timely and accurate detection of faults can enhance safety and operational efficiency significantly.

4.1 Experimental Setup

In the experimental setup, the genetic algorithm defined in Section 2.2 and 3.1 is employed to learn monitorable temporal properties that are critical for the fault detection task. The learning process is structured around three experimental cases, each designed to test various scenarios of property learning, pertaining to different splits of the trace set $\mathcal{X}_{train}$:

- *Anomaly vs. All:* in this scenario, the genetic algorithm is run separately for each type of anomaly ($\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$) related to the `MOV.m` sensor, where each run aims to learn a distinctive monitor encoded as a temporal property in bSTL that can detect the specific fault. Formally, for each anomaly type $\mathcal{A}_i$, with $i \in \{1, 2, 3\}$, $\mathcal{X}_{pos} = \mathcal{A}_i$, and $\mathcal{X}_{neg} = \mathcal{N} \cup \bigcup_{j \neq i, j \in \{1,2,3\}} \mathcal{A}_j$. This setup facilitates targeted learning for each fault type.
- *All vs. Nominal:* the algorithm runs with $\mathcal{X}_{pos} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$, encompassing all types of anomalies, and $\mathcal{X}_{neg} = \mathcal{N}$, consisting of only nominal traces. The objective is to extract a general property that differentiates any faulty condition from normal operations.
- *Anomaly vs. Nominal:* similar to the first case, this setup runs the algorithm for each anomaly type $\mathcal{A}_i$, but although, $\mathcal{X}_{pos}$ contains traces from $\mathcal{A}_i$ for all $i \in \{1, 2, 3\}$, and $\mathcal{X}_{neg}$ is exclusively composed of nominal traces in $\mathcal{N}$. This configuration is intended to refine the detection properties to distinguish specific faults directly from normal behavior.

These experimental designs are critical in validating the robustness and specificity of the learned properties, ensuring that the fault detection system is both effective in identifying various faults and efficient in differentiating these from normal operational states.

In each experimental scenario, the learning phase produces a pool of properties denoted as $\mathcal{P}$. These properties are temporal formulas derived through the genetic algorithm, designed to effectively monitor and detect faults within the system. By integrating the `rtamt` monitoring algorithm within the genetic algorithm's fitness function, the properties in $\mathcal{P}$ are designed to be runtime-verifiable. This design allows them to be evaluated on a finite prefix of each execution trace, making them suitable for online monitoring. The efficacy of these properties is subsequently evaluated during the testing phase, where their performance is critically assessed. In this stage, the algorithm as of Listing 1 applies the learned properties to the test dataset $\mathcal{X}_{\text{test}}$ to validate their accuracy and reliability, a crucial step for verifying whether the properties can correctly identify faults without excessive false positives or negatives.

Operationally, the algorithm initializes counters for nominal traces (nn), anomalous traces (np), true positives (tp), and false positives (fp) (line 2). It then iterates through each trace in the test dataset, applying the learned temporal properties to detect faults as in Def. 7 (lines 3–21), determining whether each trace is correctly identified as faulty or nominal.

Listing 1 Fault monitoring (testing phase)

```
input: non-empty pool 𝒫 of formulas, test dataset 𝒳_test
nn, np, tp, fp ← 0, 0, 0, 0
for Π ∈ 𝒳
    if LABEL(Π, len(Π) − 1) % faulty trace
        np ← np + 1
    else       % nominal trace
        nn ← nn + 1
    end if
    for i ← 0 to len(Π)-1
        Σ ← Π[0, i]
        ℱ ← { ψ ∈ 𝒫 | eb-mon(Σ, ψ) returns true }
        if ℱ ≠ empty % fault detected
            if LABEL(Π, i) % true positive
                tp ← tp + 1
            else       % false positive
                fp ← fp + 1
            end if
            break
        end if
    end for
end for
P ← tp / (tp + fp) % Calculate performance metrics, first Precision
R ← tp / np        % Recall
FAR ← fp / nn      % False Alarm Rate
F1 ← 2 * (P * R) / (P + R) % F1-score
return P, R, FAR, F1
```

The function $\textsc{label} : \mathcal{X} \times \mathbb{N} \to \{\top, \bot\}$, is defined such that $\textsc{label}(\Pi, i) = \bot$ if $\Pi \in \mathcal{N} \vee i < 150$, and $\textsc{label}(\Pi, i) = \top$ otherwise. Intuitively, this function categorizes each trace as either nominal ($\bot$) or anomalous ($\top$), facilitating the classification and evaluation of the test results. Here, 150 corresponds to the point in the signal where faults were injected, which, in a general case, may vary across different runs of the system.

The calculated metrics – precision (P), recall (R), false alarm rate (FAR), and F1-score ($F1$) – provide a comprehensive quantitative assessment of the properties' performance. Precision (P) measures the accuracy of fault detection, recall (R) assesses the algorithm's ability to identify all relevant instances, the false alarm rate (FAR) indicates the frequency of incorrect fault predictions among nominal traces, and the F1-score ($F1$) balances precision and recall in a single metric.

This detailed evaluation ensures the robustness and specificity of the learned properties, affirming the fault detection system's effectiveness and efficiency in distinguishing between normal and faulty operational states.

4.2 Experimental Results

Following the detailed description of the experimental setup, we now present and analyze the outcomes of our experiments to showcase the performance of the proposed property learning methodology. For each experimental scenario, a set of 10 trials was performed using 10 different random seeds to ensure the statistical robustness of the results. This led to 10 different pools of properties for each evaluated scenario.

The final experimental results are obtained by running the procedure from Listing 1 on the test set $\mathcal{X}_{test}$. As summarized in Table 1, the results show that the *Anomaly vs. Nominal* scenario achieved the best overall performance across all metrics, indicating a strong ability

to detect distinct types of faults compared to normal behaviors with minimal false alarms. This effectiveness is due to the targeted training approach, where properties were tuned to distinguish specific anomalies from nominal traces.

On the other hand, the *Anomaly vs. All* scenario showed lower precision but very high recall and FAR. This suggests that while the model is effective in identifying nearly all fault conditions, it also misclassifies many normal operations as faults, leading to a high rate of false alarms. This could be due to the inclusion of multiple fault types in the negative class, which can complicate and reduce the effectiveness of the learning phase, as it contains a diverse range of behaviors difficult to characterize within a single property.

The *All vs. Nominal* scenario yields intermediate performance compared to the others, with moderate values across all metrics. This outcome results from the learned properties needing to generalize across various types of anomalies, which leads to higher false alarm rates than those observed in the *Anomaly vs. Nominal* scenario.

These insights are instrumental for refining the fault detection strategies, particularly emphasizing the need for scenario-specific tuning of the learning process to enhance precision and reduce false alarms, thereby making the fault detection system more reliable and applicable in operational settings.

4.2.1 Extended Scenario: Fault Isolation

In this extended scenario, we focus on a simple fault isolation case. Specifically, this scenario advances the *Anomaly vs. Nominal* setup by considering an additional fault type, $\mathcal{A}_4$ (methane leakage), as $\mathcal{X}_{pos}$, while maintaining nominal traces in $\mathcal{X}_{neg} = \mathcal{N}$. Through this extension, the genetic algorithm was tasked with learning properties able to effectively distinguish methane leakage anomalies from normal operational states, culminating in a tailored pool of properties optimized for this specific detection challenge.

Following this learning phase, we obtained two distinct pools of properties: $\mathcal{P}_{\text{sens}}$ for sensor-related anomalies (previously learned for traces in $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ associated with `MOV.m` sensor's anomalies) and $\mathcal{P}_{\text{leak}}$ for methane leakage anomalies in $\mathcal{A}_4$. Subsequently, we embarked on an evaluative phase to assess how these properties, contained within $\mathcal{P}_{\text{sens}}$ and $\mathcal{P}_{\text{leak}}$, perform in classifying these two distinct sources of faults. For this purpose, we applied a modified version of Listing 1 to the combined property pool $\mathcal{P}' = \mathcal{P}_{\text{sens}} \cup \mathcal{P}_{\text{leak}}$ on the subset of all test traces with anomalies, $\mathcal{X}'_{\text{test}} = \{\Pi \mid \Pi \in \mathcal{X}_{\text{test}} \wedge \Pi \notin \mathcal{N}\}$.

We computed several key metrics to evaluate our performance. First, we consider *accuracy*, defined as the number of correct detections divided by the total number of traces in $\mathcal{X}'_{\text{test}}$. We also measured the rate of *missed detections* which quantifies instances where no property in $\mathcal{P}'$ is satisfied. Additionally, we assessed the rate of *overlapping detections*, which measures cases where a trace simultaneously satisfies properties from both $\mathcal{P}_{\text{sens}}$ and $\mathcal{P}_{\text{leak}}$. The experimental results from this extended scenario demonstrate a high degree of *accuracy*, with an average of 0.96 and a standard deviation of 0.03. The rate of *missed detections*, where neither type of fault was identified, averaged at 0.01 with a standard deviation of 0.01. Furthermore, the rate of *overlapping detections*, which indicates instances where both types of faults were erroneously detected simultaneously, averaged at 0.04 with a standard deviation of 0.03.

Table 1 Summary of experimental results for each testing scenario.

Scenario	Precision	Recall	FAR	F1-score
Anomaly vs. All	0.42 ± 0.00	$\mathbf{0.99 \pm 0.01}$	0.99 ± 0.01	0.60 ± 0.01
All vs. Nominal	0.71 ± 0.05	0.88 ± 0.06	0.34 ± 0.08	0.78 ± 0.05
Anomaly vs. Nominal	$\mathbf{0.92 \pm 0.05}$	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.10 \pm 0.07}$	$\mathbf{0.95 \pm 0.03}$

These metrics underscore the robustness of the proposed solution in adapting to a new fault isolation task and illustrate the model's flexibility and effectiveness in differentiating between types of faults that could occur within the system. Nevertheless, the potential of this methodology to be applied in more complex, real-world settings where multiple faults may occur simultaneously requires further validation through tailored experimentation.

5 Discussion

In this section, we delve into a comprehensive analysis of the experimental outcomes derived from our research, addressing both the strengths and limitations of our approach. This exploration aids in understanding the nuanced implications of our methodology and situates our contributions within the broader landscape of fault detection technologies.

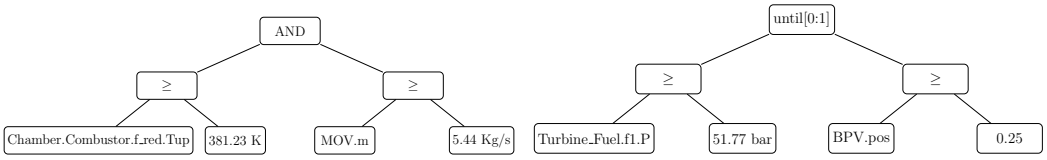
The results presented in this study underscore the efficacy of our property learning-based approach in fault detection for LPREs. Particularly in the *Anomaly vs. Nominal* scenario, our method exhibited exceptional precision and recall, highlighting its capability to effectively pinpoint specific fault conditions. Such precise detection is crucial for minimizing operational disruptions and enhancing system safety.

5.1 Strengths and Limitations of the Property Learning Approach

In the following, we present a detailed analysis of the strengths and limitations of our property learning-based approach to fault detection. This evaluation aims to comprehensively understand the potential impact and practical implications of our methodology within the field, providing a balanced view of its practical applicability in real-world settings.

The primary strength of our method lies in its high specificity and sensitivity, particularly notable in scenarios such as *Anomaly vs. Nominal*, where the method demonstrated precise fault detection capabilities. The genetic algorithm's adaptability allows it to tailor properties specifically for different fault types and operational contexts, which is evident from the high performance metrics achieved in targeted scenarios. Furthermore, the integration of the `rtamt` monitoring algorithm facilitates real-time application, a critical feature for operational settings where timely fault detection is paramount. This capability ensures that the learned properties can be applied in real-time, providing timely detections crucial for operational safety. Additionally, the interpretability of the learned properties enhances their usability, making the approach more accessible to domain experts who can understand and trust the detection logic. It is important to point out that the genetic algorithm at the core of our proposed methodology can effectively be integrated in a preemptive failure detection framework, as demonstrated in [7]. This further showcases the versatility of the approach across various operational settings.

However, the complexity of the training process forms a notable challenge. The need to encompass all potential fault scenarios in the training dataset makes the process resource-intensive and potentially cumbersome. Furthermore, our approach faces generalization challenges, as demonstrated by the elevated false alarm rates in scenarios requiring property generalization across diverse fault types. This indicates a trade-off between specificity and generalizability, highlighting the dependence on comprehensive, high-quality training data for effective property learning. Due to the extensive resources required for the property learning phase, this phase should ideally be executed offline. Once learned, monitoring of the properties can occur in real-time and can be executed even on hardware with limited resources, such as the control systems present in launch vehicles, ensuring operational feasibility in critical applications. Additionally, in all anomalous traces considered for this study, a single



(a) **(b)**

Figure 3 Syntactic trees of two properties extracted from the property learning phase representing **(a)** a `MOV.m` sensor fault and **(b)** a methane leakage.

fault is simulated while in a real-world scenario multiple faults may happen in parallel. This aspect has not been investigated, and it may affect the capability of our solution to function effectively under such conditions.

In summary, the adaptability and real-time application capabilities of our method provide significant advantages for fault detection in complex systems. However, the training complexity and challenges in generalization underscore the need for careful implementation and dataset preparation to fully realize the potential of the proposed approach.

5.2 Interpretability

Interpretability stands as a critical dimension in the assessment of fault detection methods, particularly when applying complex algorithms like genetic algorithms for property learning. The properties extracted in this study underscore not only the method's effectiveness but also its transparency, essential for practical implementation and trust by domain experts.

Figure 3 presents two representative properties derived from the pool obtained during the *Anomaly vs. Nominal* scenario, which exhibited the best performance metrics. These properties are interpreted below, demonstrating the method's ability to link observable engine behaviors to potential underlying faults, making the solution both effective and understandable.

The first property (Figure 3a) relates to the `MOV.m` sensor, crucial in measuring the mass flow of the main oxidizer valve. The condition expressed in this property asserts that if the temperature at the `Chamber.Combustor.f_red.Tup` remains above 381.23 K, while `MOV.m` is simultaneously above 5.44 Kg/s, a fault is likely occurring. This scenario suggests an anomaly where high fuel mass flow does not lead to expected increases in temperature, possibly indicating a sensor reading failure or calibration issue.

The second property (Figure 3b) concerns methane leakage, a critical safety hazard. This property's logic uses the temporal until operator, highlighting conditions that persist over a specific period. It specifies that the pressure at the `Turbine_Fuel.f1.P` should remain above 51.77 bar until the `BPV.pos` exceeds 0.25, indicating abnormal conditions likely due to a leakage affecting a pipeline or manifold close to the bypass valve.

These examples illustrate how genetic algorithms help derive actionable insights from complex data streams in real-time applications. The derived properties are not only high performing but also provide clear, interpretable logic for domain experts. This interpretability bridges the gap between automated fault detection systems and practical engineering applications, ensuring that the outputs are understandable and, hence, adoptable for practical implementation in real-world applications.

Statistics from the *Anomaly vs. Nominal* scenario further highlight the effectiveness and sensible application of the learned properties. Defined by properties with an average horizon of 5.37 (corresponding to 0.537 s in our setup) and an average height of 4.60, the monitors cover sufficient complexity to capture essential dynamics without overfitting, ensuring robustness across different operational settings, while preserving their understandability.

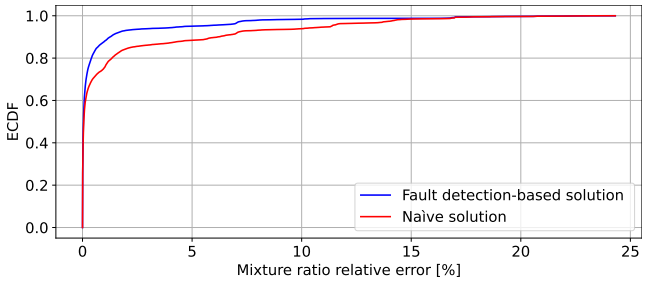


Figure 4 Empirical Cumulative Distribution Function (ECDF) of the mixture ratio relative prediction error for the naïve and fault detection-based virtual sensing solutions.

In order to enhance the intuitiveness of the generated properties, our genetic algorithm's objectives such as *parsimony*, *responsiveness*, and *timeliness* (as defined in Section 3.1) aim to limit the size and temporal horizon of the generated properties. From a structural perspective, let us point out that we can easily deduce from the STL syntax that there are at most two sub-trees for any STL operator, so that there is a direct link between an STL formula's parse tree depth and the size of the formula – both associated with a property's intuitiveness. With the objectives, we thus aim at enhancing the properties' interpretability indirectly (via their structure). Currently, we do not employ an optimization stage that would consider a user's preferences though, which could be implemented, e.g., via rewriting the properties during the generation or a posteriori. A future evaluation of the generated properties by STL engineers in terms of interpretability, and deriving corresponding optimization options (automatically or using a human in the loop) as well as evaluating their effectiveness and efficiency will help us to further improve our concept's usability aspects.

5.3 Virtual Sensing Optimization

Once a fault is detected and identified, leveraging this detection allows for the implementation of various mitigation strategies to reduce the impact caused by the fault. One such strategy involves the use of a virtual sensing model to replace faulty measurements.

We demonstrate this capability on an estimator for the mixture ratio – the ratio of oxidizer to fuel within the combustion chamber – a critical factor in maintaining optimal combustion efficiency and engine performance. This quantity is hard to measure directly due the dynamic conditions during rocket operation. The XGBoost regressor [10] used for the estimation is trained on nominal training data, $\mathcal{X}_{\text{train}} \cap \mathcal{N}$, ensuring that it learns from fault-free operational conditions, as discussed in [34].

In our experiment, two approaches were evaluated: (*i*) *Naïve*, where the estimator utilizes test data in $\mathcal{X}_{test}$ that includes both faulty sensor readings and nominal runs without any adjustments, providing a baseline performance measure; (*ii*) *Fault Detection-Based*, which involves feeding the estimator with test data in $\mathcal{X}_{test}$ that has been adjusted based on real-time monitoring of the engine's operational traces, following the steps outlined in Listing 1. Specifically, when a fault is detected by the monitoring algorithm, mitigation strategies are applied within the `if` branch (lines 12–19) for the remaining timesteps of the run. In this scenario, `MOV.m` sensor values are estimated using a virtual sensing model (another XGBRegressor) that has been previously trained exclusively on the partition of nominal training data $\mathcal{X}_{\text{train}} \cap \mathcal{N}$. This approach aims to correct sensor data anomalies before they are used in mixture ratio estimations, thereby enhancing the accuracy of predictions.

The effectiveness of solution (*ii*) is supported by Figure 4, which compares the empirical cumulative distribution functions (ECDF) of the mixture ratio prediction relative errors for both (*i*) *Naïve* and (*ii*) *Fault Detection-Based* approaches. The *Fault Detection-Based* solution exhibited a clear improvement in prediction accuracy over the *Naïve* one, highlighting the benefits of integrating real-time fault detection with virtual sensing.

6 Related Work

Detecting anomalous behavior in CPS is a vibrant research domain. Machine learning and deep learning are featured prominently in this field, due to their efficacy, even though the resultant black-box models lack interpretability. Recent work highlights neural networks which learn temporal relationships in the data in order to detect faults and failures in cloud data centers [19], aero-propulsion systems [2], disk drives [2, 22] and electrocardiogram (ECG) data [28]. To foster explainability and ensure that solutions are applicable to different domains and contexts, recent approaches combining machine learning and formal techniques have emerged. Specifically, [25, 6, 9] present STL property mining techniques, using genetic algorithms, decision trees, and reinforcement learning to distinguish between different time series data. Bartocci et al [4] provide a survey of 15 procedures covering template-based and template-free, model-based and model-free, active and passive, and supervised and unsupervised methods. According to their categorization, the method presented in here is template-free, model-free, passive, and supervised, i.e. learning from labeled data.

Diagnosis in CPS is an established field that focuses on identifying the root causes of system malfunctions, which is crucial for maintaining system reliability and safety. Notably, [27] has contributed significantly to ML-based diagnosis, highlighting techniques that combine machine learning with model-based approaches for effective monitoring and diagnosability of CPS. Additionally, [11] has explored diagnostic methods specifically for assessing the diagnosability of systems, ensuring that potential faults can be accurately detected and isolated during system operation. Furthermore, [24] demonstrates the application of classifier-based approaches in real-time fault detection scenarios. Unlike these cited works, which primarily address diagnosability and model-based diagnosis, our approach integrates property learning directly into the monitoring process, enabling both real-time fault detection and subsequent diagnosis in a unified process.

7 Conclusions

In this work, we tackled the challenge of developing a multi-objective genetic programming methodology that automatically learns STL properties for fault detection in LPREs, effectively distinguishing between nominal and faulty behaviors. This methodology was validated with a comprehensive dataset simulating various fault conditions, ensuring the robustness and accuracy of the learned properties.

The experimental results demonstrated the effectiveness of the proposed approach in learning STL properties essential for run-time fault detection in LPRE control systems. The high precision and recall in distinguishing normal from faulty behaviors ensure timely and accurate fault detection, which is crucial for maintaining LPRE operational integrity. Moreover, incorporating virtual sensing, guided by detected faults, significantly improved the accuracy of critical parameter estimations, such as the combustion chamber's mixture ratio, underscoring our solution's practical applicability. The interpretability of the learned STL properties further enhances our approach's value, making it both effective and accessible to domain experts, thereby facilitating adoption in real-world aerospace applications.

Future research avenues include developing models for nominal engine behavior to enable (unsupervised) anomaly detection, evaluating cases with simultaneous faults in multiple sensors to assess and improve the robustness in complex scenarios, and devising strategies to minimize redundant physical sensors ensuring the operational functionality of the engine.

References

1 A. Abid, M. T. Khan, and J. Iqbal. A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review*, 54(5):3639–3664, 2021. doi:10.1007/S10462-020-09934-2.

2 K. Aggarwal, O. Atan, A. K. Farahat, C. Zhang, K. Ristovski, and C. Gupta. Two birds with one network: Unifying failure event prediction and time-to-failure modeling. In *Proc. 6th Big Data*, pages 1308–1317. IEEE, 2018. doi:10.1109/BIGDATA.2018.8622431.

3 B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

4 E. Bartocci, C. Mateis, E. Nesterini, and D. Nickovic. Survey on mining signal temporal logic specifications. *Information and Computation*, 289:104957, 2022. doi:10.1016/J.IC.2022.104957.

5 A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010. doi:10.1093/LOGCOM/EXN075.

6 G. Bombara and C. Belta. Offline and online learning of signal temporal logic formulae using decision trees. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–23, 2021. doi:10.1145/3433994.

7 A. Brunello, D. Della Monica, A. Montanari, N. Saccomanno, and A. Urgolo. Monitors that learn from failures: Pairing STL and genetic programming. *IEEE Access*, 11:57349–57364, 2023. doi:10.1109/ACCESS.2023.3277620.

8 Y. Cao, B. J. Smucker, and T. J. Robinson. On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design. *Journal of Statistical Planning and Inference*, 160:60–74, 2015.

9 G. Chen, M. Liu, and Z. Kong. Temporal-logic-based semantic fault diagnosis with time-series data from industrial internet of things. *IEEE Transactions on Industrial Electronics*, 68(5):4393–4403, 2020. doi:10.1109/TIE.2020.2984976.

10 T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. 22nd SIGKDD*, pages 785–794. ACM, 2016.

11 A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *Proc. 18th IJCAI*, pages 363–369, 2003.

12 E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. *Model checking*. MIT Press, 2nd edition, 2018.

13 K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2013. doi:10.1109/TEVC.2013.2281535.

14 A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *Computer Aided Verification*, pages 264–279. Springer, 2013. doi:10.1007/978-3-642-39799-8_19.

15 K. Dresia, M. Boerner, W. Armbruster, S. Klein, T. Traudt, D. Suslov, J. Hardi, G. Waxenegger-Wilfing, and J. Deeken. Design and Control Challenges for the LUMEN LOX/LNG Expander-Bleed Rocket Engine. In *Proc. 34th ISTS*, 2023.

16 A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

17 European Commission. Communication from the commission to the european parliament, the european council, the council, the european economic and social committee and the committee of the regions- the european green deal. Technical report,

European Commission, 2019. URL: https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0002.02/DOC_1&format=PDF.

18 F. A. Fortin, F. M. De Rainville, M. A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012. doi:10.5555/2503308.2503311.

19 J. Gao, H. Wang, and H. Shen. Task failure prediction in cloud data centers using deep learning. *IEEE Transactions on Services Computing*, 15(3):1411–1422, May 2022. doi:10.1109/TSC.2020.2993728.

20 J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

21 M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. doi:10.1016/J.JLAP.2008.08.004.

22 S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi. Making disk failure predictions smarter! In *Proc. 18th FAST*, pages 151–167. USENIX, 2020.

23 O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

24 I. Matei, J. de Kleer, A. Feldman, M. Zhenirovskyy, and R. Rai. Classification based diagnosis: Integrating partial knowledge of the physical system. In *Proc. 11th PHM*, 2019.

25 L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi. A robust genetic algorithm for learning temporal specifications from data. In *Proc. 15th QEST*, pages 323–338. Springer, 2018. doi:10.1007/978-3-319-99154-2_20.

26 D. Ničković and T. Yamaguchi. RTAMT: Online robustness monitors from STL. In *Proc. 18th ATVA*, volume 12302, pages 564–571. Springer, 2020. doi:10.1007/978-3-030-59152-6_34.

27 O. Niggemann, G. Biswas, J. S. Kinnebrew, H. Khorasgani, S. Volgmann, and A. Bunte. Data-driven monitoring of cyber-physical systems leveraging on big data and the internet-of-things for diagnosis and control. In *Proc. 26th DX*, pages 185–192, 2015.

28 G. Petmezas, K. Haris, L. Stefanopoulos, V. Kilintzis, A. Tzavelis, J. A. Rogers, A. K. Katsaggelos, and N. Maglaveras. Automated atrial fibrillation detection using a hybrid CNN-LSTM network on imbalanced ECG datasets. *Biomedical Signal Processing and Control*, 63:102194, 2021. doi:10.1016/J.BSPC.2020.102194.

29 I. Pill and F. Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139:124–141, 2018. doi:10.1016/J.JSS.2018.02.002.

30 A. Pnueli. The temporal logic of programs. In *Proc. 18th SFCS*, pages 46–57. IEEE, 1977.

31 S. Pérez-Roca, J. Marzat, H. Piet-Lahanier, N. Langlois, F. Farago, M. Galeotta, and S. Le Gonidec. A survey of automatic control methods for liquid-propellant rocket engines. *Progress in Aerospace Sciences*, 107:63–84, 2019.

32 V. S. Reddy. The SpaceX effect. *New Space*, 6(2):125–134, 2018.

33 T. Traudt, W. Armbruster, C. Groll, R. Hahn, K. Dresia, M. Börner, S. Klein, D. I. Suslov, J. Haemisch, M. A. Müller, J. Deeken, J. Hardi, and S. Schlechtriem. LUMEN, the Test Bed for Rocket Engine Components: Results of the Acceptance Tests and Overview on the Engine Test Preparation. In *Proc. 9th 3AF Space Propulsion*, 2024.

34 A. Urgolo, C. Gei, K. Dresia, M. Freiberger, E. Kurudzija, H. Neumann, I. Pill, F. Pittino, G. Radchenko, and G. Waxenegger-Wilfing. Feature selection and virtual sensing based mixture ratio estimation for liquid propellant rocket engine control systems. In *Proc. 9th 3AF Space Propulsion*, 2024.

Quantifying the Sim-To-Real Gap in UAV Disturbance Rejection

Austin Coursey
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Abstract

Due to the safety risks and training sample inefficiency, it is often preferred to develop controllers in simulation. However, minor differences between the simulation and the real world can cause a significant sim-to-real gap. This gap can reduce the effectiveness of the developed controller. In this paper, we examine a case study of transferring an octorotor reinforcement learning controller from simulation to the real world. First, we quantify the effectiveness of the real-world transfer by examining safety metrics. We find that although there is a noticeable (around 100%) increase in deviation in real flights, this deviation may not be considered unsafe, as it will be within $>$ 2m safety corridors. Then, we estimate the densities of the measurement distributions and compare the Jensen-Shannon divergences of simulated and real measurements. From this, we show that the vehicle's orientation is significantly different between simulated and real flights. We attribute this to a different flight mode in real flights where the vehicle turns to face the next waypoint. We also find that the reinforcement learning controller actions appear to correctly counteract disturbance forces. Then, we analyze the errors of a measurement autoencoder and state transition model neural network applied to real data. We find that these models further reinforce the difference between the simulated and real attitude control, showing the errors directly on the flight paths. Finally, we discuss important lessons learned in the sim-to-real transfer of our controller.

2012 ACM Subject Classification Computing methodologies → Control methods; Computing methodologies → Model development and analysis

Keywords and phrases sim-to-real, disturbance rejection, unmanned aerial vehicles

Digital Object Identifier 10.4230/OASIcs.DX.2024.16

Funding This work was supported by NASA award #80NSSC21M0087-21-S06.

Acknowledgements We want to thank Luis Alvarez at MIT Lincoln Laboratory for his help in running UAV flights.

1 Introduction

Reinforcement learning (RL) techniques have shown a wide range of success across a variety of tasks such as games [22], robotic hand manipulation [2], and racing simulations [28]. In the realm of diagnosis, this class of algorithms has been used to perform fault diagnosis of rotating machinery [9, 18] and to reduce data imbalance in fault diagnosis [10]. RL has also been used to increase the resilience of dynamical systems. For example, a recent study covers the application of RL in enhancing the reliability of power and energy systems [11]. RL methods are classified into dynamic response optimization, restoration and recovery, energy management, cybersecurity, and resilient planning. In unmanned aerial vehicles (UAVs), RL has been used to avoid collisions [16, 4] and increase safety in windy [30, 6] and fault conditions [23].

© Austin Coursey, Marcos Quinones-Grueiro, and Gautam Biswas;
licensed under Creative Commons License CC-BY 4.0

Novel RL methods hold great potential for real-world applications. However, due to the sample inefficiency of RL algorithms [31], they often cannot be trained in the real world. Additionally, RL-based agents can make dangerous decisions during the learning process that can risk the safety of the system [19]. Therefore, it is often desired to train these agents in simulation.

Inaccuracies in the simulation model used to train RL methods define the existing *sim-to-real gap* [26, 7]. This gap can impact the performance of controllers when transferred to the real world, leading to reduced performance [19]. While techniques such as domain randomization [26] have been developed to reduce this gap during simulation, methods for quantifying the gap and isolating causes for worse real-world performance are understudied. In this paper, we examine a case study of the transfer of an octorotor UAV disturbance rejection controller from simulation to real flights. We detail three methods for examining the sim-to-real gap. First, we quantify the overall safety of the real system and compare it to simulation using a set of safety metrics. Then, we estimate the density distributions of the measurements and determine which measurements failed to match between the simulation and the real world using their Jensen-Shannon divergence. Finally, we learn an Autoencoder neural network and system transition neural network on simulated data to determine which points along the UAV flight had a larger sim-to-real gap.

In this paper, we make the following contributions.

1. We show a recent case study of the transfer of an octorotor UAV disturbance rejection controller from simulation to the real world. This case study gives evidence for the potential of the combination of RL agents as supervisory controllers and widely used classic controllers to maintain UAV safety in the real world.
2. We study the potential of three methods for examining and isolating the sim-to-real gap. These are (1) system safety metrics, (2) measurement distribution analysis, and (3) neural network transfer error inspection.
3. We discuss important lessons we learned in transferring from simulation to the real world. These lessons can help future researchers ensure a more successful transfer of UAV disturbance rejection controllers.

The rest of the paper is structured as follows. In Section 2, we summarize related research on this topic. In Section 3, we give brief preliminary definitions. In Section 4, we detail the controller development process and its transfer to the real world. In Section 5, we present our methods for quantifying and identifying the sim-to-real gap. In Section 6, we examine the sim-to-real gap on our octorotor case study. In Section 7, we discuss important lessons we learned through this case study. Finally, in Section 8, we conclude the paper.

2 Related Work

Transferring a UAV controller from simulation to the real world is a common task in control literature. For example, in [14], a sigma-pi neural network is used to compensate for unknown dynamics in a PID control architecture. This algorithm is applied to a real quadrotor. In [12], they compare feedback control methods to a nonlinear model predictive control method for quadrotors in simulation. Then, they applied the algorithms to a real drone. They compared the performance of these algorithms in the real world. In [29], they design a controller for quadrotor disturbance rejection. They use this controller with a real quadrotor in an outdoor flight test. In [15], a deep reinforcement learning controller that directly controls the motor speeds of a UAV is developed. They apply their algorithm to a real hexacopter UAV. Although there is a wide range of literature that applies control algorithms developed in a simulator to the real world, most of these papers do not focus on methods for determining the success of this transfer.

Instead, much of the recent literature focuses on techniques for improving the sim-to-real transfer. For example, the sim-to-real gap has been studied in UAV computer vision tasks. In [8], they compare the drone detection performance in simulated and real-world settings. They attempt to match the environmental conditions in simulation to real videos. They use vision-specific metrics like the mean average precision to compare drone detection performance in simulation and the real world. In [24], they attempt to mitigate the sim-to-real gap by integrating computer vision into their autonomous landing algorithm. To reduce the sim-to-real gap, they replicate the appearance of real-world landing pads in their simulator. In [17], they present a framework for developing aerial robotics systems while avoiding the sim-to-real gap. Their framework involved developing the system in simulation, integrating the algorithm with the hardware, performing validation in augmented reality, and finally using the system in an industrial application. They apply this framework for UAV wind turbine inspection. In [27], they use domain randomization techniques to improve the sim-to-real transfer of a fixed-wing UAV control algorithm in a wind tunnel. They showed that domain randomization improves real-world performance. Techniques like these offer promising ideas for improving the transferability of algorithms developed for UAVs in simulation. However, the evaluation of the quality of the sim-to-real transfer is limited to qualitative flight profiles or state error computations.

Outside of UAVs, perhaps the most active area of sim-to-real research is on techniques for reducing the gap at simulation time. Domain randomization [26] is one such popular approach. In domain randomization, some parameters of the simulation are randomized during the model learning process. By learning to generalize across these parameters, the model should learn to generalize to the real world better. This, along with transfer learning, is widely successful in robotics tasks [19]. To address the fact that the randomized parameters need to be known beforehand, [13] proposes an approach that automates parameter tuning. They learn a causal graph to determine which environmental parameters lead to a trajectory difference. In [25], they improve the sim-to-real transfer of their robot by using system identification techniques to improve the accuracy of their simulation. Then, they improve the robustness of the controller with domain randomization, random perturbations, and reducing the observation space. Again, these techniques do not focus on quantifying the sim-to-real gap. Some recent papers do [5], but the evaluation is limited to inspecting high-level errors instead of focusing on understanding why the gap exists.

In this paper, we present a case study of the transfer of an octocopter UAV reinforcement learning disturbance rejection controller from simulation to the real world. Our analysis goes beyond state error metrics, comparing the distributions of metrics and leveraging the learning process of neural networks to isolate unexpected behaviors.

3 Preliminaries

In this paper, we attempt to quantify the *sim-to-real gap*. The sim-to-real gap, or reality gap, can be simply defined as a discrepancy between simulated and real-world environments [26, 7]. By reducing the gap, controllers or models developed in simulation can be better applied in real systems.

The controller we developed in simulation is a reinforcement learning UAV disturbance rejection controller. Reinforcement learning aims to find a solution to a sequential decision-making problem expressed as a Markov Decision Process (MDP). An MDP consists of $\mathcal{S}$, a set of environmental states, $\mathcal{A}$, the actions the agent can take in the environment, p, a transition function $p(s'|s,a)$ that denotes the probability of reaching a next state, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

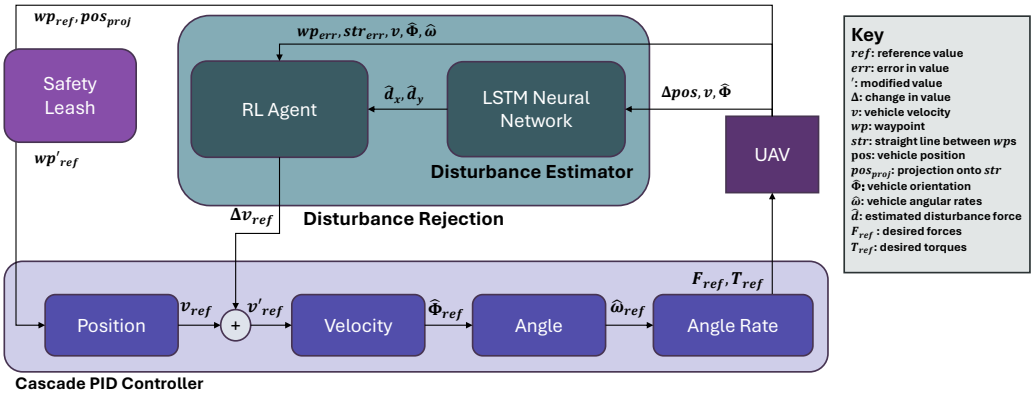


Figure 1 Full UAV disturbance rejection controller. Figure and controller modified from [6] to include the estimated disturbance forces as an RL state.

a reward function for taking an action in a state. Therefore, the MDP $= \{\mathcal{S}, \mathcal{A}, p, r\}$. The solution to this problem is characterized by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected discounted future reward.

4 Controller Development and Transfer

This paper discusses transferring a hybrid reinforcement learning and control-based UAV controller to the real world. The goal of this controller is to perform disturbance rejection. The controller should increase the safety and robustness of the UAV under disturbances. The controller was primarily developed with strong wind in mind. This section details the controller development process in simulation and its application on a real UAV.

4.1 Simulation

Before using any controllers on a real UAV, we developed a disturbance rejection controller in simulation. We modeled and simulated a Tarot T-18 octocopter UAV. This UAV model consisted of 8 18-inch propellers, was about 1.25 meters in diameter, and had a 10.66 kg mass. To simulate this UAV, we used an open-source Python multirotor UAV simulator [1]. To simulate flights, we provided a predefined set of reference waypoints the UAV needed to reach by getting within a 2-meter radius. In simulation, the UAV was allowed to fly in a 2-meter cylindrical corridor between waypoints, the safety corridor. In the real world, we may consider less strict constraints, such as with a 5-meter safety corridor. Outside this corridor, there could be obstacles that compromise the safety of the UAV.

A common UAV control scheme is a cascade proportional-integral-derivative controller. This control scheme stacks PID controllers to go from a reference position to reference forces and torques to be applied to the UAV. Despite its widespread use, it is a linear control scheme with limited robustness to strong wind conditions [6]. See Figure 1 for an illustration of this control scheme. To move beyond the rigidness of this controller while still maintaining its strengths as a stable and trusted control algorithm, we integrated it with a deep reinforcement learning agent.

To develop the agent, we considered a set of features that would be reliably observable at 2 Hz or faster on a real UAV. These made up the state space. The state space consisted of 17 features, the distance to the next waypoint in meters $wp_{err} \in \mathbb{R}^3$, the vehicle inertial velocity

$v \in \mathbb{R}^3$, the angular orientation $\hat{\Phi} = [\phi, \theta, \psi] \in [-\pi, \pi]^3$, the angular rates $\hat{\omega} \in \mathbb{R}^3$, the distance to the straight line between waypoints $str_{err} \in \mathbb{R}^3$, and the estimated disturbance forces in the x and y directions $Fx \in \mathbb{R}, Fy \in \mathbb{R}$. To estimate the disturbance forces, we trained a long short-term memory (LSTM) neural network model on data from simulated flights. This LSTM used data from the past second (sampled at 0.25 seconds for real flights and 0.1 seconds for real flights). It used the full-second time window of the change in position, the vehicle velocity, and the orientation as input to predict the disturbance X and Y forces.

The agent was allowed to modify the X and Y reference velocities in the cascade PID controller. These modifications are added to the reference velocity given by the position controller (see Figure 1). The actions were restricted to modify the reference velocity by at most 1 meter a second. Therefore, the action space was $\Delta v_{ref} \in [-1, 1]^2$. Additionally, the agent was constrained to only provide actions every 0.5 seconds and receive state observations every 0.25 seconds to mimic the delay between a computer on a ground station and a UAV. The actions were applied for the full half a second until the next decision.

To train the agent, we used the proximal policy optimization (PPO) algorithm [20]. We trained the agent to maximize the following reward function.

$$\text{reward}(\text{state}_t) = -\frac{\text{tte}(pos, wp_{\text{prev}}, wp_{\text{ref}})}{T_{\text{safe}}} + \text{bonus}_t \tag{1}$$

$$\text{tte}(pos, wp_{\text{prev}}, wp_{\text{ref}}) = \frac{\|(pos - wp_{\text{prev}}) \times (wp_{\text{ref}} - wp_{\text{prev}})\|}{\|(wp_{\text{ref}} - wp_{\text{prev}})\|} \tag{2}$$

In this equation, pos is the current position, wp_{prev} is the previous waypoint, wp_{ref} is the desired waypoint, T_{safe} is a normalization constant (the radius of the safety corridor), and the trajectory tracking error (tte) is the deviation from the straight line between waypoints. Additionally, there is a bonus for completing a mission or a penalty for failing. We trained the agent over a series of simulated flights in a square trajectory. The agent experienced random constant wind vectors with turbulence each flight. The controller was commanded to have a fixed yaw of 0 radians. For full details of this controller along with a rule-based waypoint modification scheme we designed, please refer to our previous work where we developed and evaluated a previous version of this controller strictly in simulation [6]. The only differences in the controller used for this paper are that it estimates the disturbance forces instead of the wind speed and can only make small X and Y velocity adjustments.

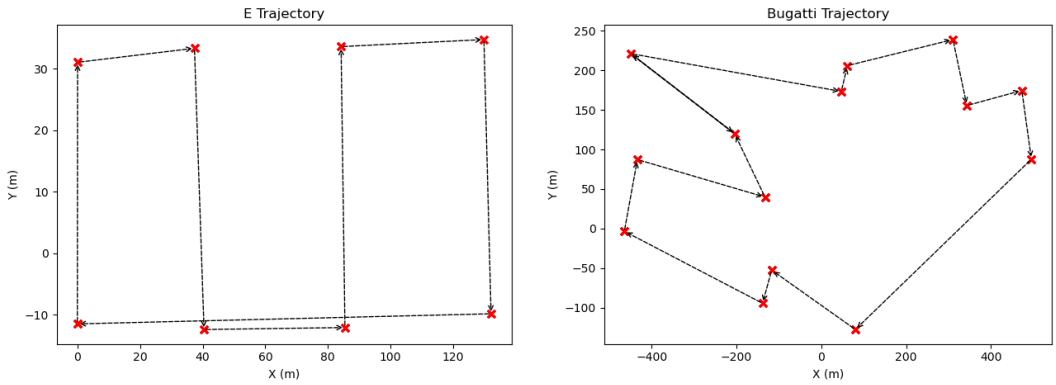
With the problem of sim-to-real transfer in mind, we took a few steps to increase the transfer effectiveness during this phase. First, we used domain randomization [26] during the training process. In domain randomization, some properties of the simulation are varied across trials. In our case, we varied the wind direction, the wind magnitude, and the flight direction in each episode. Intuitively, if an agent can learn a general policy across variations in the simulation, it may generalize better to the real world. Second, we added noise to the observations to mimic measurement noise. Third, we varied the orientation of the rotors on the UAV to empirically determine whether the UAV could generalize to model variations in simulation. Finally, we obtained real octocopter flight data from NASA Langley and qualitatively validated the realism of our measurements.

4.2 Real-World

We integrated our controller with the autopilot control of a DJI S100 octocopter UAV through a framework that uses the Mavlink protocol for data exchange. The airframe of the UAV has a diameter of about 1.25m and weighs around 10 kg, similar to the T-18 used for simulation. The octocopter has 8 18-inch propellers. A picture of the drone can be seen in Figure 2.



Figure 2 Image of the octocopter we used for the real flight experiments.



(a) E shape trajectory.

(b) 'Bugatti' trajectory.

Figure 3 Desired trajectories for the real flights. The desired altitude was constant.

The UAV was controlled through the ArduPilot software. To connect our controller to this software, a ground station computer used our position controller to compute reference velocities every 0.25 seconds, our data sampling rate. Then, those reference velocities were sent to ArduPilot to perform low-level control of the UAV. Every 0.5 seconds, our RL controller, also on a ground station computer, modified the reference velocities to counteract any estimated disturbances. Notably, this control scheme is slightly different than our simulation scheme. In simulation, we use a cascade PID scheme, described in the previous section. On the real UAV, we used the autopilot software. Although we modeled our cascade UAV after the controllers used in this software, we did not directly use it when developing our controller. This likely led to some differences in behavior in the real flights. Additionally, this software commanded the UAV to face its next waypoint when flying toward it instead of facing completely straight (it used yaw control). In simulation, we did not use this operating mode. The impacts of this change in operating mode are studied extensively in Section 6.

We flew 5 full flights using our controller at Fort Devens, Massachusetts. The trajectories the UAV flew can be seen in Figure 3. We flew a shorter E trajectory that flew in all cardinal directions and a longer "bugatti" trajectory with more complex turns.

5 Methods for Quantifying the Sim-to-Real Gap

With our real UAV flights wrapped up, we wanted to determine how well the controller we developed in simulation transferred to the real world. To do so we replicated the real flights in simulation, compared the overall safety of flights, analyzed the distributions of features, and leveraged the learning process of neural network models to uncover differences in real and simulated flight data.

To directly compare the real flights to simulated ones, we replicated the real flights in simulation. For two of the flight days, we made the UAV hold its altitude with no X or Y position commands. We took the average vehicle velocity during this phase as an estimate of the wind vector for that flight. This left us with 5 full flights (4 E shape and 1 bugatti shape), with known wind conditions. Then, we ran those flights in simulation, adding Dryden turbulence to the wind to increase the realism.

5.1 Safety Metrics

A simple way to quantify the sim-to-real gap in UAVs is to compare important flight-level metrics. In this case, since it is a disturbance rejection controller that aims to improve octorotor UAV safety, we considered the following metrics. In the computation of these metrics, we used the trajectory tracking error (Equation 2) and notated its input as a measurement vector, where M is the set of all measurement vectors in that flight, to simplify notation.

- **Mean Deviation (Mean TTE)**. The average distance from the straight line between waypoints across the flight. If the mean deviation is smaller than the radius of the safety corridor, that means the UAV was safe on average.

$$\frac{1}{|M|} \sum_{m_t \in M} \text{tte}(m_t) \tag{3}$$

- **Maximum Deviation (Max TTE)**. The maximum distance from the straight line between waypoints across the flight. If the maximum deviation is smaller than the radius of the safety corridor, then that means the UAV never violated safety in that flight.

$$\max(\{\text{tte}(m_t) \mid m_t \in M\}) \tag{4}$$

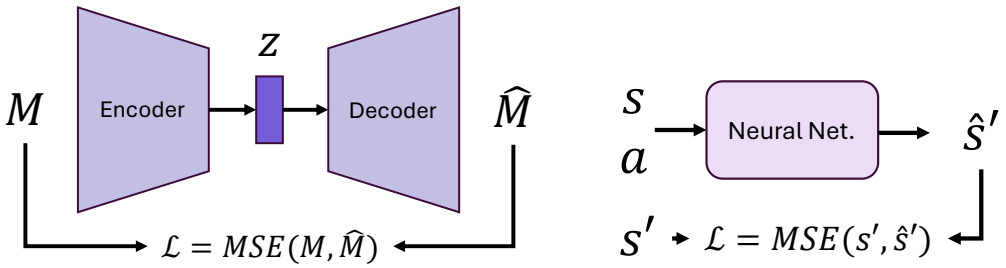
- **Percent of Time Outside Safety Corridor (% TOC)**. The percentage of the flight spent outside a predefined safety corridor. In the following equation, r is the radius of the safety corridor.

$$\frac{|\{m_t \in M \mid \text{tte}(m_t) > r\}|}{|M|} \tag{5}$$

5.2 Measurement Distributions

Next, we quantified the sim-to-real gap for individual measurements. However, since the simulated and real measurements are not aligned in time or space we cannot directly use the difference in measurements as a fair comparison. Instead, we compared the distributions of measurements.

To model the distribution of measurements, we performed kernel density estimation using the statsmodels Python package [21]. Then, with a continuous distribution for each measurement, we compared the distributions using the Jensen-Shannon divergence (JSD),



(a) Measurement autoencoder. **(b)** State transition model neural network.

Figure 4 Neural network architectures used to quantify the sim-to-real gap. The measurement autoencoder will fail to reconstruct any differences in real-world measurements. The state transition model will predict an incorrect next state if there is a large sim-to-real gap.

shown in the equations below where $\mathcal{X}$ is a common sample space and P and Q are the estimated density distributions.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{6}$$

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2} D_{\mathrm{KL}}(P \parallel \frac{1}{2}(P+Q)) + \frac{1}{2} D_{\mathrm{KL}}(Q \parallel \frac{1}{2}(P+Q)) \tag{7}$$

A JSD close to 0 implies the two distributions are almost identical, and a JSD closer to $\ln(2) \approx 0.693$ implies the two distributions are completely different.

5.3 Leveraging Neural Networks

While measurement distribution analysis can give an overall sense of the sim-to-real gap, it does not allow for easy inspection of the specific scenarios where the sim-to-real transfer failed or was successful. To do so, we trained two neural network models, a measurement autoencoder and a state transition model.

5.3.1 Measurement Autoencoder

The first neural network we trained to quantify the sim-to-real gap was a measurement Autoencoder. We used a vanilla neural network autoencoder instead of a variational autoencoder or another variant because we do not care about the distribution of the latent space; we only care that it serves as a bottleneck. This Autoencoder, shown in Figure 4, compresses the measurements M down to a latent vector z using an encoder neural network. Then, this latent vector is projected back to the measurement space using a decoder neural network. It was trained to minimize the mean squared error of the measurement reconstructions, shown below where n is the dimension of the measurement vector M.

$$MSE(M, \hat{M}) = \frac{1}{n} \sum_{i=1}^{n} (M_i - \hat{M}_i)^2 \tag{8}$$

We trained this Autoencoder on two simulated flights (one Bugatti and one E), leaving a final E flight for validation. The measurements we considered were the error to the next waypoint, the vehicle's velocities, the vehicle's angular orientation, and the vehicle's angular

Table 1 Optimal hyperparameters. The same parameters were given to the encoder and decoder of the Autoencoder. Notice that the transition model trained for longer with a higher learning rate and a larger network, implying this may be a more difficult task.

Model	Epochs	Learning Rate	Batch Size	Layers	Neurons	Latent Dim.
Autoencoder	24	0.00060	16	2	256	6
Transition	44	0.00098	16	3	256	-

rate. These made up a 12-dimensional input space. We normalized this data and optimized the hyperparameters on the validation data using the tree-structured Parzen estimator algorithm [3]. The optimal hyperparameters are shown in Table 1.

Although the task of learning to reconstruct the input seems trivial, the compression of data to a latent vector loses information, keeping the Autoencoder from being an identity function. Intuitively, the reconstruction error should be high when applying this model to real data if there is a large sim-to-real gap. Additionally, we can inspect which measurements are causing this error and map these errors to flight paths.

5.3.2 State Transition Model

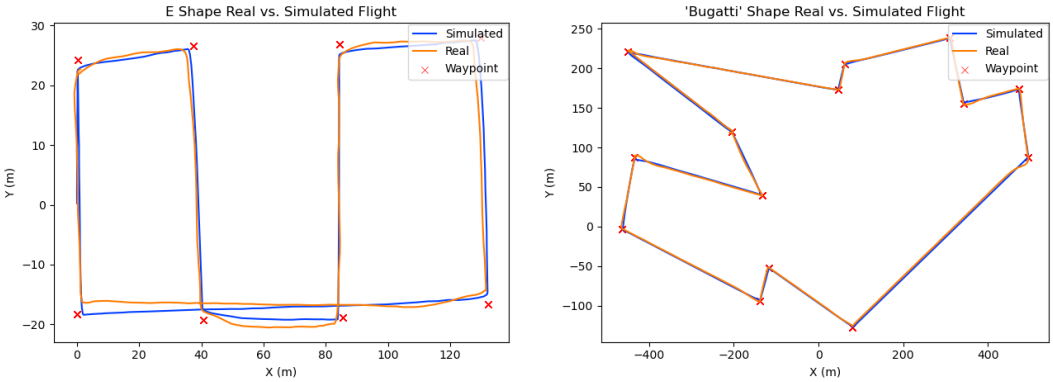
Next, we trained a neural network to mimic a system transition model. This neural network, shown in Figure 4, predicts the next state from the current state and action. It was trained to minimize the mean squared error of the predicted next state. The states were the same measurements as for the state Autoencoder above. The actions were the actions of the reinforcement learning agent, the change in reference X and Y velocity. Therefore, this neural network models the transition from the perspective of the reinforcement learning agent.

We trained this agent with the same training process as the Autoencoder network, optimizing the hyperparameters on a simulated validation flight and training on the other simulated flights. The optimal hyperparameters are shown in Table 1. By training on the simulated flights, it learns the simulation transition function. If there is a large sim-to-real gap, this network should incorrectly predict the next state on real-world data.

6 Results

First, we can analyze the sim-to-real gap in terms of the overall system safety. Figure 5 shows a qualitative comparison of the real and simulated flight paths. On the shorter E trajectory, we can see minor differences in the behavior in the real world. Notably, the simulated flight seems to be straighter. The real UAV slightly overshot the waypoint near (80, -20) and undershot the waypoint near (0, -20). Additionally, the flights seemed less stable overall in the real world. This difference appears more prominent when flying in the X direction, but this is likely due to the scale of the axes. In the Bugatti trajectory, the differences between the real and simulation seem even smaller. The only clear difference is minor waypoint overshooting, like at (-400, 100), in the real flight.

We can quantify the difference between these simulated and real flights by looking at safety metrics. Table 2 shows these metrics. From this table, we can see that there is a clear difference between the performance of the real and the performance of the simulated flights. In both the E shape and bugatti shape flight paths, the average deviation (Mean TTE) was about half in the simulation. The same pattern holds for the maximum deviation and total deviation. This indicates that there is a sim-to-real gap. However, this gap may not



(a) E flight path. Wind of (1, 0.5) m/s.

(b) 'Bugatti' flight path. Wind of (-3.5, -2.5) m/s.

Figure 5 Real and simulated flight paths. Note the difference in axis scales between the E and Bugatti trajectories.

Table 2 Safety metrics for real and simulated flights. In the E shape category, the mean of the 4 real E flights and the mean of the 2 simulated E flights are shown. The time outside the corridor metrics (TOC) are notated with the radius of the safe corridor.

Type	Mean TTE (m)	Max TTE (m)	2m TOC (%)	5m TOC (%)
		E Shape		
Real	1.180	4.617	15.9	0
Simulated	0.491	1.996	0	0
		Bugatti Shape		
Real	1.286	4.987	20.7	0
Simulated	0.544	2.042	0.2	0

have been large enough to impact the safety of the flights. None of the flights left a 5-meter corridor around the straight line between waypoints. Additionally, the real flights were within a 2-meter corridor on average, leaving at most 20% of the time during the Bugatti flight. Depending on the specific safety requirements, the quality of the sim-to-real transfer may be sufficient for these trajectories.

6.1 Distributional Analysis

From the safety metric analysis, we determined that there was a noticeable difference in performance in real flights. To isolate the cause of this difference, we can start by analyzing the measurement distributions. Figure 6 shows the kernel density estimations of the measurement distributions. It also shows the Jensen-Shannon divergence (JSD) between the distributions. For this trajectory, many of the measurements have similar distributions. For example, the X and Y positions and velocities have small JSD. Additionally, the difference in the deviation distributions reinforces the superior performance of the controller in simulation. There is a large difference in the yaw distributions. In the real flights, we were unable to fix the yaw at 0 radians as we did in the simulation. This is a dramatic change in the UAV behavior. (The UAV turned after reaching each waypoint instead of always facing forward.) By noticing the large JSD, we can isolate this change in behavior. Aside from the yaw, the disturbance force

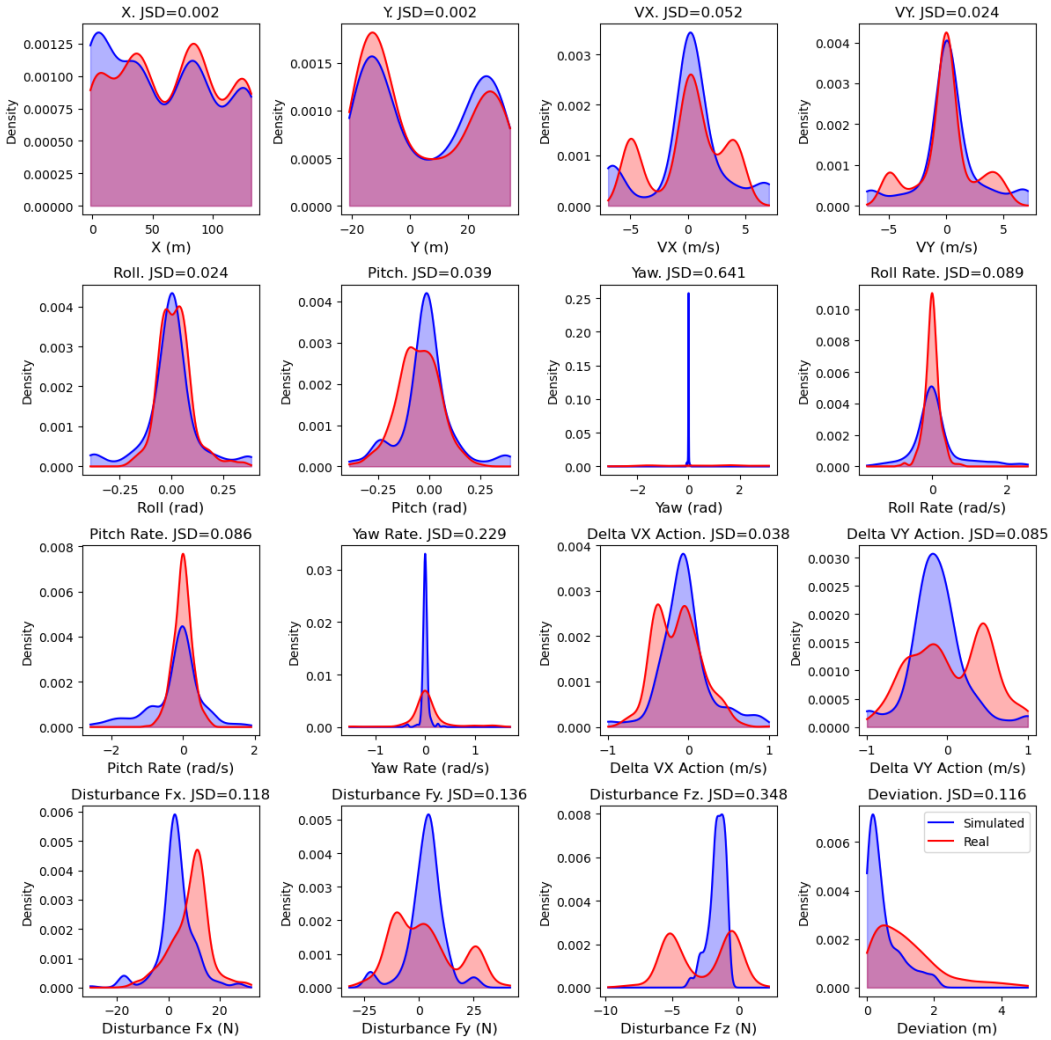


Figure 6 Kernel density estimations and Jensen-Shannon divergences between real and simulated flight distributions for the E-shape flights. The disturbance forces are estimated by the disturbance estimation neural network in the controller.

estimations given by the controller had large divergences. These estimations are in the UAV body frame. Since the heading of the UAV changed in the real flights, the directions of the disturbances in the body frame perspective changed. Therefore, the change in yaw likely caused this behavior difference too.

Figure 7 shows similar results for the Bugatti shape flight. In this flight, we see larger deviations in the roll and pitch than the E shape, likely due to the more complex turns and longer stretches with differing yaw angles. The deviation from the straight line between waypoints in the simulated flights was multimodal for this trajectory. The second mode being centered around a deviation of 1 meter implies that this trajectory was more difficult.

To determine the impact of the change in heading on the disturbance estimator, we converted the estimated disturbances from the vehicle body frame to the inertial frame. Figure 8 shows these results for the E shape trajectory. The estimated disturbance forces for the simulated flights closely matched the prescribed wind vector (1, 0.5) while the disturbance

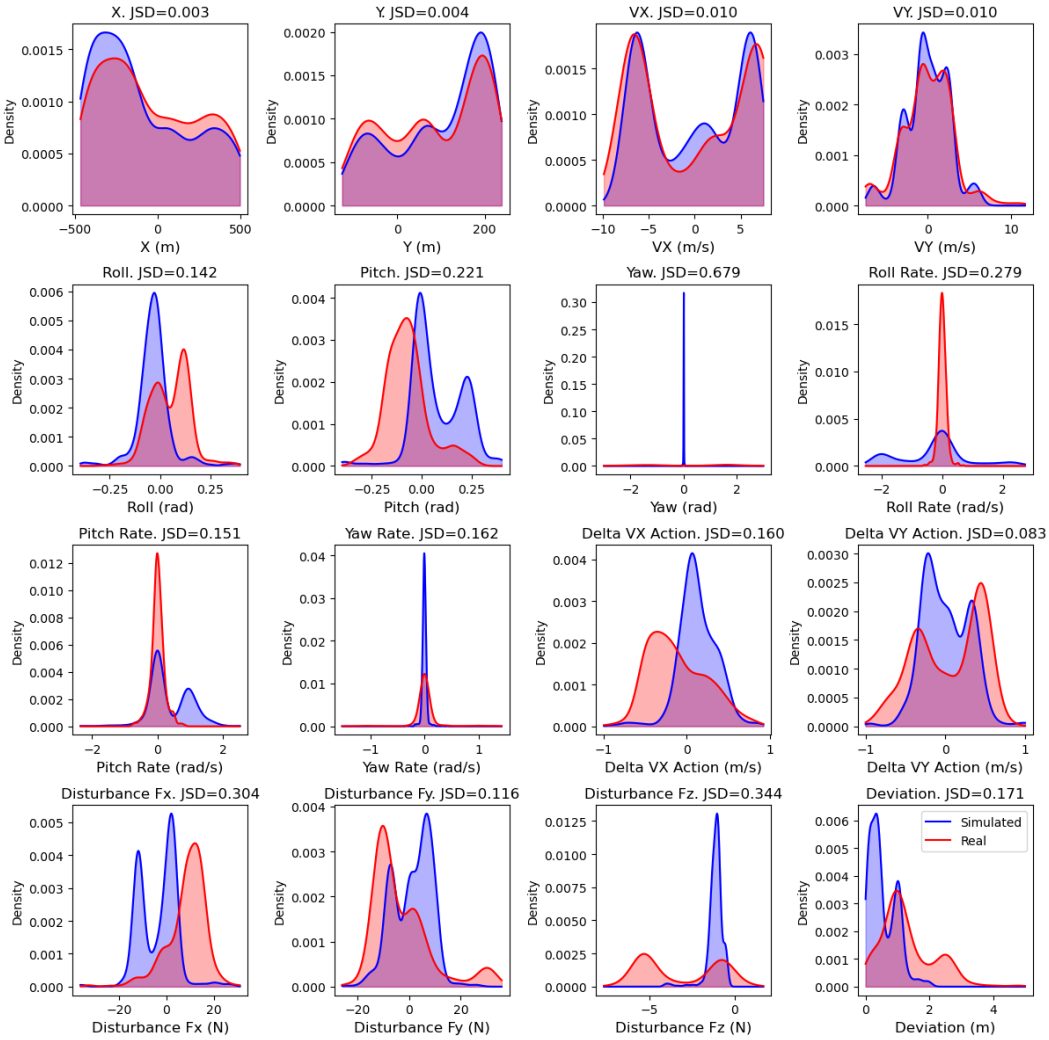
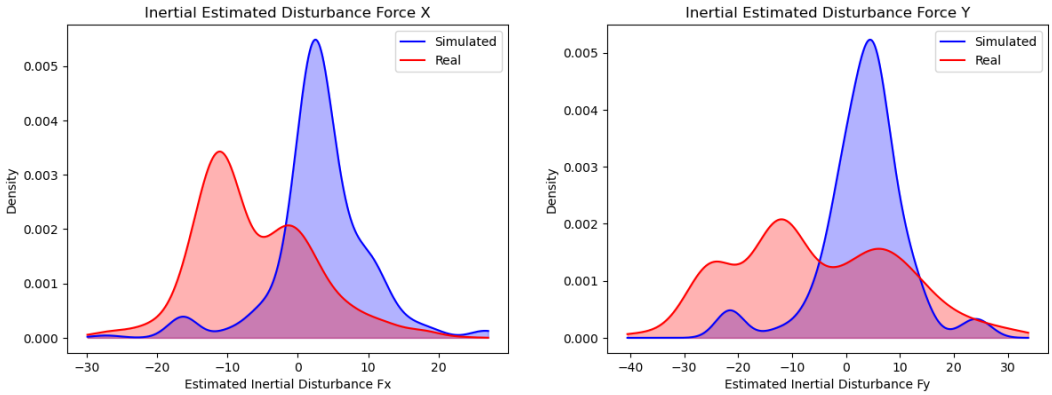


Figure 7 Kernel density estimations and Jensen-Shannon divergences (JSD) between real and simulated flight distributions for the Bugatti-shape flights. The disturbance forces are estimated by the disturbance estimation neural network in the controller.

forces for the real flights did not. As previously mentioned, we estimated the wind vector after the flight, so the disturbance forces shown in Figure 8 may accurately represent the true disturbance forces during the flight. However, this difference may instead be caused by the disturbance estimator failing to generalize to real flights with a different operating behavior (heading towards the waypoint) than in simulation.

In both Figure 6 and Figure 7, we see evidence that the controller is operating as intended. The actions given during the real flights by the agent (Delta VX Action and Delta VY Action in the figures) were skewed in the opposite direction as the estimated disturbance forces. This shows the agent attempted to properly reject disturbances in the real world. Despite the sim-to-real gap in the angle control, the controller appeared to make proper decisions.



(a) Inertial estimated disturbance force X. **(b)** Inertial estimated disturbance force Y.

Figure 8 Estimated disturbance forces output by the disturbance estimator converted to the inertial frame to account for differences in yaw control between simulated and real flights. Example shown for E shape trajectory. The simulated disturbances forces closely follow the estimated wind vector. The real estimated disturbance forces are much nosier and do not always match the wind.

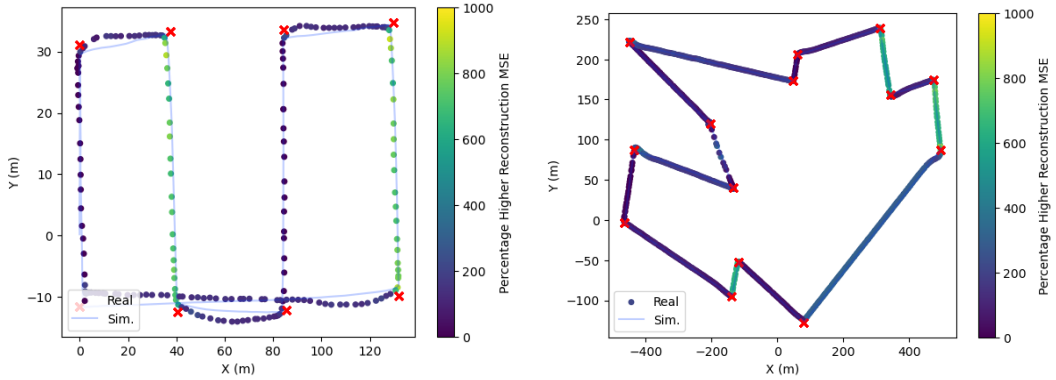
6.2 Quantifying the Gap along Flight Paths

Although these density estimations highlight important sim-to-real differences on a distribution level, they do not map these issues to the flight path and potentially miss subtle behavior differences. To do so, we can inspect the Autoencoder reconstruction error along the flights. This is shown in Figure 9. In this figure, the highest reconstruction errors are shown when the UAV is flying south. This is when the yaw is most different. When flying north, the reconstruction errors are very low. Additionally, the simulated flight reconstruction error is not higher when flying south. This indicates that the sim-to-real gap is largely caused by the difference in flight mode.

Figure 10 shows the autoencoder reconstruction errors without considering yaw or the yaw rate as measurements. Without these features, the reconstruction errors were much lower. This indicates that the other states more closely match between the simulation and the real world. The patterns in the reconstruction errors of these features are less obvious from the flight paths, indicating errors may be coming from more subtle differences or from secondary features that were impacted by the change in heading in real flights.

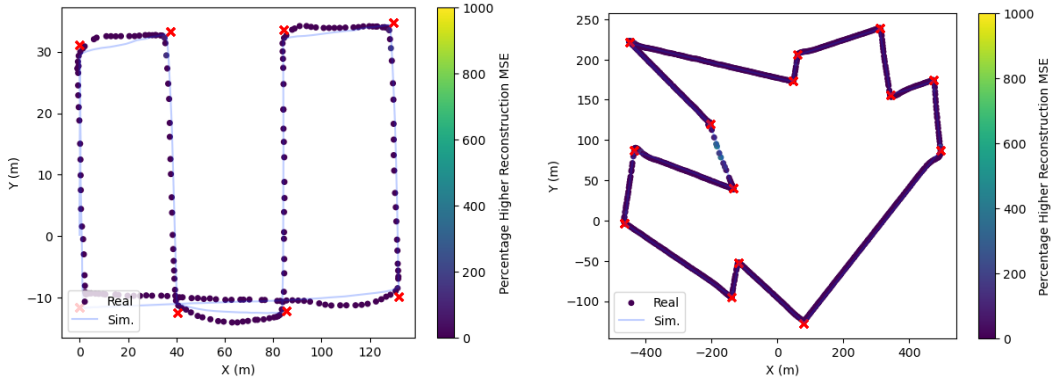
Finally, Figure 11 shows the errors of the system transition neural network along the flight paths. This quantifies the sim-to-real gap from the perspective of the reinforcement learning agent. If the error is higher, the system transition model may be different, and we may not be able to expect the controller to make consistent decisions. In this figure, we see that the largest errors are again when flying south. This may be a cause for the difference in safety metrics between simulated and real flights. However, the UAV does not deviate significantly, even while flying south. This means that even though there was a difference in the system transition, the agent may have been partially robust to it. Since the agent was trained with wind gusts, it sometimes experienced different yaw angles than 0 radians. Its generalizability to this scenario may be attributed to this.

These results indicate a clear next step. The controller needs to be expanded to support a yaw-changing operating mode or the autopilot on the real drone needs to be modified to force the yaw commands at 0 radians. Through our methods for sim-to-real quantification and visualization, we were able to isolate this issue and demonstrate its importance. We also



(a) E flight path. **(b)** 'Bugatti' flight path.

Figure 9 Real flight paths colored by the percentage higher the real autoencoder reconstruction error was. A higher error by the autoencoder indicates a sim-to-real gap. Notice the main source of error comes when flying south, where the UAV faces the opposite direction as in the simulation.



(a) E flight path. **(b)** 'Bugatti' flight path.

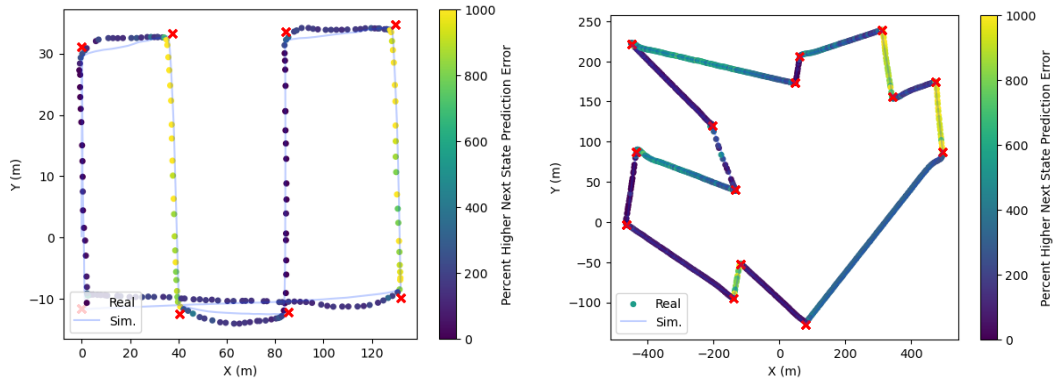
Figure 10 Real flight paths colored by the autoencoder reconstruction error without considering yaw or the yaw rate. Notice that the errors are much smaller and less frequent. This implies fixing this controller difference may greatly improve the sim-to-real transfer.

found that this may not have significantly degraded the performance of the controller, as it still makes correct decisions and maintains an average deviation within a 2-meter safety corridor.

7 Discussion and Lessons Learned

Through this case study, we identified key points that may help future researchers as they transfer UAV controllers from simulations to the real world.

- **We can quantitatively isolate and identify issues in the sim-to-real gap.** In this paper, we demonstrated four methods to quantify the sim-to-real gap. Three of those, the kernel density estimations, measurement autoencoder, and state transition neural network, were able to isolate specific issues in the transfer to the real world. By calculating the divergence between measurement distributions, we could identify problematic features. To identify differences in behavior in the real world, we mapped the errors of the neural network models onto the flight paths. This can lead to faster and more accurate controller improvements.



(a) E flight path.

(b) 'Bugatti' flight path.

Figure 11 Real flight paths colored by the state transition neural network error. A higher error indicates that the model predicted a different next state than was observed in the real world.

- **It is important to make reasonable assumptions when developing a controller in simulation.** When developing our controller, we assumed we could fix the desired UAV yaw to 0 radians on a real drone. However, this assumption did not hold. By avoiding restrictive assumptions like this and building flexibility into the controller in simulation, the sim-to-real gap can be lessened.
- **Despite the sim-to-real gap, controller performance may be sufficient.** We quantitatively identified issues in the transfer of our controller from simulation to the real world. However, the practical impacts of this sim-to-real gap may be small. The UAV was able to flight within a safe zone on average and the qualitative flight paths are reasonable. Further decreasing the sim-to-real gap may not be worth the time and cost of controller iteration.
- **We can develop controllers with the sim-to-real gap in mind.** We attribute the success of our controller in the real world to the techniques we used when developing the controller, namely randomizing parameters of the simulation during controller training and evaluation. By performing domain randomization, the agent may generalize to parameters of the real world. We also performed a comprehensive analysis of our controller's performance in simulation, trying to ensure that we limited the number of unexpected scenarios. In the real world, we commanded the UAV to hold its altitude at the end of flights so that we could replicate the wind conditions in simulation.

8 Conclusion

In this paper, we analyzed a case study on the transfer effectiveness of a UAV disturbance rejection controller from simulation to the real world. We developed a reinforcement learning controller that modifies the reference velocity of a cascade PID control scheme. This agent was trained and evaluated completely in simulation. Then, we flew real octocopter flights using this controller. Next, we replicated those flights in simulation. Using this data, we qualitatively inspected the difference in flight profiles, observing that the simulated UAV flew straighter and overshot waypoints less. Then, we compared the safety of simulated and real flights, finding that real flights deviated from a 2-meter safety corridor around 16% of the time while the simulated flights never left the corridor. Despite this difference, the safety of the real flights is adequate for many applications, such as ones with larger

corridors. To isolate which features have the largest sim-to-real gap, we performed kernel density estimation on the measurements. We calculated the Jensen-Shannon divergence of those densities, finding that the yaw had the largest divergence due to the real UAV turning when reaching a waypoint. We also used these densities to show that the controller actions counteracted the estimated disturbances, implying the controller produced correct actions in the real world. Then, we used an autoencoder neural network model to learn to reconstruct simulated measurements. We mapped its reconstruction errors on real data onto flight paths to show that the largest sim-to-real gap was when flying south. Additionally, we trained another neural network to learn the state transition model to show that the transition when flying south was also different, potentially reducing controller performance. Finally, we discussed important lessons learned through this case study.

Future work should apply these sim-to-real quantification metrics on other systems to determine their generalizability across diagnosis domains. It should also test this controller in more extreme wind conditions in the real world to quantify the sim-to-real gap in settings where dire reinforcement learning intervention is needed. We hope that research in this area will improve the efficiency of transfer from simulation to the real world and the overall safety of real systems.

References

1 Ibrahim Ahmed, Marcos Quinones-Grueiro, and Gautam Biswas. A high-fidelity simulation test-bed for fault-tolerant octo-rotor control using reinforcement learning. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*. IEEE, 2022.

2 OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. doi:10.1177/0278364919887447.

3 James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

4 Omar Bouhamed, Hakim Ghazzai, Hichem Besbes, and Yehia Massoud. Autonomous uav navigation: A ddpg-based deep reinforcement learning approach. In *2020 IEEE International Symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2020. doi:10.1109/ISCAS45731.2020.9181245.

5 Jack Collins, David Howard, and Jurgen Leitner. Quantifying the reality gap in robotic manipulation tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6706–6712. IEEE, 2019.

6 Austin Coursey, Allan Zhang, Marcos Quinones-Grueiro, and Gautam Biswas. Hybrid control framework of uavs under varying wind and payload conditions. In *2024 American Control Conference (ACC)*. IFAC, 2024.

7 Iván García Daza, Rubén Izquierdo, Luis Miguel Martínez, Ola Benderius, and David Fernández Llorca. Sim-to-real transfer and reality gap modeling in model predictive control for autonomous driving. *Applied Intelligence*, 53(10):12719–12735, 2023. doi:10.1007/S10489-022-04148-1.

8 Tamara Regina Dieter, Andreas Weinmann, Stefan Jäger, and Eva Brucherseifer. Quantifying the simulation–reality gap for deep learning-based drone detection. *Electronics*, 12(10):2197, 2023.

9 Yu Ding, Liang Ma, Jian Ma, Mingliang Suo, Laifa Tao, Yujie Cheng, and Chen Lu. Intelligent fault diagnosis for rotating machinery using deep q-network based health state classification: A deep reinforcement learning approach. *Advanced Engineering Informatics*, 42:100977, 2019. doi:10.1016/J.AEI.2019.100977.

10 Saite Fan, Xinmin Zhang, and Zhihuan Song. Imbalanced sample selection with deep reinforcement learning for fault diagnosis. *IEEE Transactions on Industrial Informatics*, 18(4):2518–2527, 2021. doi:10.1109/TII.2021.3100284.

11 Mukesh Gautam. Deep reinforcement learning for resilient power and energy systems: Progress, prospects, and future avenues. *Electricity*, 4(4):336–380, 2023.

12 Bryan S Guevara, Luis F Recalde, José Varela-Aldás, Victor H Andaluz, Daniel C Gandolfo, and Juan M Toibero. A comparative study between nmpc and baseline feedback controllers for uav trajectory tracking. *Drones*, 7(2):144, 2023.

13 Peide Huang, Xilun Zhang, Ziang Cao, Shiqi Liu, Mengdi Xu, Wenhao Ding, Jonathan Francis, Bingqing Chen, and Ding Zhao. What went wrong? closing the sim-to-real gap via differentiable causal discovery. In *Conference on Robot Learning*, pages 734–760. PMLR, 2023. URL: https://proceedings.mlr.press/v229/huang23c.html.

14 Fan Jiang, Farhad Pourpanah, and Qi Hao. Design, implementation, and evaluation of a neural-network-based quadcopter uav system. *IEEE Transactions on Industrial Electronics*, 67(3):2076–2085, 2019. doi:10.1109/TIE.2019.2905808.

15 Bodi Ma, Zhenbao Liu, Qingqing Dang, Wen Zhao, Jingyan Wang, Yao Cheng, and Zhirong Yuan. Deep reinforcement learning of uav tracking control under wind disturbances environments. *IEEE Transactions on Instrumentation and Measurement*, 72:1–13, 2023. doi:10.1109/TIM.2023.3265741.

16 Sihem Ouahouah, Miloud Bagaa, Jonathan Prados-Garzon, and Tarik Taleb. Deep-reinforcement-learning-based collision avoidance in uav environment. *IEEE Internet of Things Journal*, 9(6):4015–4030, 2021. doi:10.1109/JIOT.2021.3118949.

17 Rafael Perez-Segui, Pedro Arias-Perez, Javier Melero-Deza, Miguel Fernandez-Cortizas, David Perez-Saura, and Pascual Campoy. Bridging the gap between simulation and real autonomous uav flights in industrial applications. *Aerospace*, 10(9):814, 2023.

18 Gensheng Qian and Jingquan Liu. Development of deep reinforcement learning-based fault diagnosis method for rotating machinery in nuclear power plants. *Progress in Nuclear Energy*, 152:104401, 2022.

19 Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153187, 2021. doi:10.1109/ACCESS.2021.3126658.

20 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. arXiv:1707.06347.

21 Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

22 David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

23 Yves Sohège, Marcos Quiñones-Grueiro, and Gregory Provan. A novel hybrid approach for fault-tolerant control of uavs based on robust reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10719–10725. IEEE, 2021.

24 Charalambos Soteriou, Christos Kyrkou, and Panayiotis S Kolios. Closing the sim-to-real gap: Enhancing autonomous precision landing of uavs with detection-informed deep reinforcement learning. In *International Conference on Deep Learning Theory and Applications*, pages 176–190. Springer, 2024.

25 Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018. arXiv:1804.10332.

26 Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world.

In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. doi:10.1109/IROS.2017.8202133.

27 Daichi Wada, Sergio Araujo-Estrada, and Shane Windsor. Sim-to-real transfer for fixed-wing uncrewed aerial vehicle: pitch control by high-fidelity modelling and domain randomization. *IEEE Robotics and Automation Letters*, 7(4):11735–11742, 2022. doi:10.1109/LRA.2022.3205442.

28 Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022. doi:10.1038/S41586-021-04357-7.

29 Lin-Xing Xu, Hong-Jun Ma, Dong Guo, An-Huan Xie, and Da-Lei Song. Backstepping sliding-mode and cascade active disturbance rejection control for a quadrotor uav. *IEEE/ASME Transactions on Mechatronics*, 25(6):2743–2753, 2020.

30 Xiaolin Xu MS and Jeffrey Sun. A new trajectory in uav safety: Leveraging reinforcement learning for distance maintenance under wind variations. *Journal of Aviation/Aerospace Education & Research*, 33(4):6, 2024.

31 Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020. doi:10.1109/SSCI47803.2020.9308468.

Real-Time Sensor Fault Detection in Drones: A Correlation-Based Algorithmic Approach

Inbal Roshanski
Department of Software and Information Systems Engineering, Ben Gurion University of the Negev, Beersheba, Israel

Magenya Roshanski
CTO, Maris, Israel

Meir Kalech
Department of Software and Information Systems Engineering, Ben Gurion University of the Negev, Beersheba, Israel

Abstract

Drones, or unmanned aerial vehicles (UAVs), are becoming increasingly vital across various industries, where their reliable operation is crucial for safety and efficiency. Ensuring this reliability requires the early detection of sensor-related faults, which are critical for maintaining the performance and safety of UAVs. This study addresses this challenge by leveraging real-world data from an Aero-Sentinel Military UAV Sentinel G2 quadcopter. The data was collected through a collaboration with Maris-Tech Ltd, using their advanced Mercury Nano system to capture detailed communication between the drone and its control unit. A set of correlation-based algorithms was developed and evaluated, specifically tailored to address the unique complexities of drone sensor data, which is often influenced by environmental factors. Among the algorithms tested, two novel methods emerged as particularly effective, demonstrating significant improvement compared to previous methods, in fault detection accuracy. These methods, designed to accurately identify and predict sensor malfunctions, offer a robust solution for enhancing the reliability and safety of UAV operations.

2012 ACM Subject Classification Hardware → Failure prediction; Hardware → Sensors and actuators; Computing methodologies → Modeling methodologies

Keywords and phrases Drones, Sensor Fault Detection, Correlation-Based Algorithms, Sensor Data Analysis, Anomaly Detection, Data-Driven Fault Detection

Digital Object Identifier 10.4230/OASIcs.DX.2024.17

1 Introduction

Drones, also known as unmanned aerial vehicles (UAVs), have seen a significant rise in their utilization across various industries such as agriculture, construction, and transportation. They are employed for an array of tasks, including aerial surveillance, data collection, and cargo delivery, functioning effectively in diverse environments ranging from remote areas to densely populated urban centers. Despite their growing adoption and versatility, drones are susceptible to faults, which can result in accidents and potential damage to the equipment. This makes fault detection and prediction a critical area of research in the realm of UAV technology.

Drones rely on data collected from a multitude of sensors, such as cameras, GPS, accelerometers, and barometers. This sensor data is necessary for controlling the drone's movements and executing various tasks. Typically, the data collected by these sensors is in the form of time-series, meaning it is recorded over time at varying frequencies. Analyzing time-series data presents significant challenges due to its inherent noisiness, irregularity, and the potential presence of gaps, often caused by network failures or other technical issues.


© Inbal Roshanski, Magenya Roshanski, and Meir Kalech;
licensed under Creative Commons License CC-BY 4.0





Figure 1 Aero-Sentinel Military UAV Sentinel G2 quadcopter drone.

Drones are vulnerable to several types of faults, including sensor malfunctions, structural failures, and communication errors within the control unit. These faults can be triggered by numerous factors, such as adverse weather conditions, mechanical wear and tear, or human error. Identifying and predicting these faults before they manifest is crucial to preventing accidents and ensuring the safety of both the drone and the surrounding environment.

Although many studies have been conducted on drone fault prediction, few have addressed the challenge of detecting different types of faults while accounting for data from a wide array of sensors sampled at varying frequencies. For instance, works such as [5] and [14] focus primarily on individual sensor faults or specific fault types, without considering the complexities introduced by multiple sensors. Additionally, most existing research, such as [10] and [5], primarily involves experiments conducted using simulated data or controlled laboratory environments. This leaves a gap in evaluating fault detection methods under real-world conditions, where diverse sensor inputs and environmental factors come into play. One of the significant advancements of our research is addressing situations with real-world conditions. We conduct our experiments on data collected from actual drones operating in real-world environments, providing a more practical and robust evaluation of fault detection methodologies, particularly focusing on sensor-related faults.

A crucial aspect of our research is the collaboration with Maris [1], a company specializing in Intelligent Video Surveillance and Analytics. Maris, a pioneer in high-performance AI edge video and analytics technologies, played a key role in enabling the collection of real-world data. Utilizing their Mercury Nano system [2] – an advanced, dual-channel low-power encoder designed to integrate with a wide range of platforms – we recorded communication between an Aero-Sentinel Military UAV Sentinel G2 quadcopter drone [3] and its control unit, as shown in Figure 1. This process captured detailed sensor data influenced by various environmental factors, resulting in a dataset that encompasses approximately 17 hours of flight data collected under diverse conditions.

Our first contribution lies in the creation and presentation of this unique dataset derived from real-world drone flights. Unlike simulated or laboratory-generated data commonly used in existing studies, our dataset includes genuine environmental noise and other real-world

[1] https://www.maris-tech.com/
[2] https://www.maris-tech.com/solutions/mercury/
[3] https://www.aero-sentinel.com/military-drones/military_drone_sentinel_g2/

variables, offering a comprehensive and authentic foundation for evaluating and validating fault detection models. This dataset addresses a significant gap in the literature by reflecting the complexities inherent in drone operations.

Our second contribution is the development of a reliable fault prediction model capable of real-time detection of sensor failures in operational drones. To achieve this, we present a sensor-based approach for Fault Detection specifically tailored for drones. This approach builds upon the methodology initially introduced in [11], which combines data-driven and model-based techniques to enhance fault detection capabilities. We have adapted and refined this methodology to address the unique challenges associated with drone operations by implementing both the basic Sensor Fault Detection and Diagnosis (Simple-SFDD) and an extended version that incorporates additional features for improved performance (Extended-SFDD). In addition to these, we have developed two more versions of our own. The Combined-SFDD version combines elements of both the basic and Extended-SFDD methods, aiming to leverage the strengths of each. The Knowledge-Based-SFDD version introduces a novel variation of this algorithm, which integrates modeling to learn and predict the normal behavioral patterns of drones through data-driven analysis. These enhancements collectively contribute to more accurate and efficient detection of sensor-related faults in drone systems.

Our experiments on this dataset demonstrate the robustness and effectiveness of our approach. The results indicate that the fourth variation of our algorithm, which is based on sensor behavior learning, achieves notable improvements. Specifically, this variation shows superior performance in both True Positive Rate (TPR) and False Positive Rate (FPR), effectively bridging the gap in fault detection accuracy.

2 Related Work

Unmanned Aerial Vehicles (UAVs) are aircraft that operate without a human pilot, crew, or passengers on board. UAVs can be controlled remotely by a human operator, referred to as Remotely Piloted Aircraft (RPA), or they can operate with varying levels of autonomy, ranging from autopilot assistance to fully autonomous flight with no human intervention. UAVs are categorized based on various criteria, including the number of propellers, their configurations, and construction types, which include rotary wing, tilt-rotor, fixed wing, and flapping wing designs. Our study focuses specifically on quadrotor drones, which are UAVs equipped with four rotors and are commonly used due to their stability and maneuverability [19]. Drones, a subset of UAVs, are versatile, robot-like aircraft that can range in size from as large as a full-scale aircraft to as small as the palm of your hand.

Drones are now employed in a wide range of roles and responsibilities [3]. Over time, drones have evolved to serve various purposes, including monitoring climate change, conducting search and rescue operations during natural disasters, photography, cinematography, and delivering commodities. Drones are also heavily involved in research that addresses diverse tasks such as path planning, communication networks, autonomous control, and more [18]. These tasks are critical for enhancing drone capabilities, making them more intelligent, but they also increase the complexity of both their software and hardware – thereby making them more susceptible to faults. Given that a drone malfunction today can have serious consequences, the importance of fault detection has become paramount.

The topic of fault prediction and detection is vast and crucial for maintaining operational systems. It aids professionals in managing components or systems and even in prioritizing testing procedures. In the field of UAVs, several studies have endeavored to address this

challenge. UAV failures are typically classified into two main categories: actuator faults and sensor faults. Although there have been studies focused on damage to circuit boards [21], the drone's frame [2], or the surface of the flying vehicle itself [1], these areas are not the primary focus of most research. In our study, we will concentrate specifically on the challenge of identifying faults using the drones' sensors.
The most current survey on the topic was conducted by Puchalski et al., who categorize the available methodologies in the field into two groups [16], model-based and data-driven.

2.1 Model-Based

Model-based methods involve mathematical models that describe and predict the behavior of a system using a set of equations and algorithms. These models are developed based on assumptions and hypotheses derived from expert knowledge, and they are commonly employed to simulate and assess the system's performance under various scenarios [20]. Currently, model-based methods are the most widely used approach for detecting faults in unmanned aerial vehicles.
Herdjunanto et al. [9] were among the first to apply a model-based fault detection technique to an unmanned quadrotor. Their goal was to address actuator fault signal isolation during hovering motion. Their work focuses on isolating actuator fault signals using a detection filter, capable of managing various fault types without the need for adjusting filter settings. By modeling the quadrotor and simulating specific fault signals, they demonstrate effective fault detection through a virtual actuator approach.
The Kalman filter is a mathematical algorithm for estimating system states from noisy observations. Classified as a model-based method, the Kalman filter employs a mathematical model to describe and predict system behavior, making it a valuable tool in UAV fault detection research. [13]. Several studies have utilized the Kalman filter for this purpose. For instance, [22] applied an adaptive two-stage extended Kalman filter to detect sensor faults in UAVs, successfully modeling quadrotor kinematics and testing various fault scenarios using the Quanser Qball-X4 model. Their results confirmed the filter's effectiveness. The extended Kalman filter has also been utilized by [23] and [6] for similar applications.

In their study, Zhong et al. [23] proposed an actuator fault detection and diagnosis method specifically for quadrotors. This method is based on a linearized dynamic UAV model and a decomposed adaptive augmented state Kalman filter. Through simulation experiments, they demonstrated the effectiveness of their approach in accurately detecting actuator faults. Similarly, Demircan et al. [6] explored the capability of a nonlinear extended Kalman filter to detect aileron locking in fixed-wing aircraft. To estimate the state of the roll rate, which is directly influenced by the aileron, they employed an extended Kalman filter and conducted simulations using MATLAB. Their results indicated that the faults were successfully detected. Further advancing the use of Kalman filtering in UAV fault detection, Hamadi et al. [8] applied both the basic and extended versions of the Kalman filter to address faults in quadrotors. Their approach targeted faults caused by hardware sensor issues (e.g., GPS, IMU, and magnetometer) as well as software errors, including faults within the Kalman filters themselves or incorrect parameter settings.

Fu et al. [7] introduced a singular Markov switching system for detecting sensor and actuator faults in quadrotors. They developed an adaptive observer and validated their approach through simulations.
Maqsood et al. [14] proposed a system for detecting faults in angular rate sensors. Their approach predicts faults using a chain differentiator integrator and enhances detection accuracy with a modified high-gain observer, incorporating a sliding mode effect to improve

detection and reduce overshoot and chattering. They simulated both gradual and abrupt faults, comparing their method to traditional detection techniques. The results showed a clear advantage over other nonlinear strategies.

The Model-Based approach, while effective, faces several challenges that limit its reliability and practical application. One major issue is its reliance on expert knowledge for model building and validation, which can be difficult or impossible to obtain in some cases, reducing the model's usefulness. Additionally, the complexity of the mathematical models involved makes them challenging to understand, interpret, and implement. These models often require specialized software and expertise, making the Model-Based approach both time-consuming and difficult to apply in practice.

2.2 Data-Driven

These models are constructed using data rather than relying on expert knowledge. Typically, machine learning or statistical methods are employed to automatically uncover patterns and correlations within the data. Data-driven models offer several advantages over model-based approaches. They can capture complex and nonlinear patterns that may be challenging or impossible for model-based methods to represent. Additionally, data-driven models can evolve and improve over time as new data becomes available, thereby enhancing the model's accuracy and reliability [15].

Chen et al. [5] proposed a data-driven model using a backpropagation neural network (BPNN) optimized by a genetic algorithm. They trained the model with pitch rate data from the speed gyroscope during UAV flight. The sensor signal was first divided into eight frequency bands using a three-layer wavelet packet, and the energy feature vector of each band served as input to the neural network. Their MATLAB simulations, which tested various sensor faults, demonstrated that this method outperformed the standard BPNN in terms of accuracy and error. Continuing their research, Chen et al. [4] introduced an enhanced sensor fault detection method using the same wavelet packet and BPNN framework, but with adaptive fireworks to improve local search capability and algorithm convergence. This approach also utilized a distributed mechanism for parallel information sharing and multi-scale analysis via wavelet transform. Compared to the initial method, the adaptive fireworks algorithm showed superior classification efficiency, faster convergence, shorter runtime, and improved global search ability.

Iannace et al. [10] employed sound analysis to detect faults in quadrotor propeller blades. A microphone placed 1.2 meters from the drone captured the rotor noise, with strips of paper tape attached to the blade surface to simulate faults. Frequency analysis was then used to extract 31 features from the recorded sounds, which were used to train a feed-forward multilayer neural network. The authors reported high accuracy in fault detection, though the method's main limitation is that it can only be tested in indoor conditions.

Sadhu et al. [17] developed a method for real-time fault detection and identification using deep Convolutional and Long Short-Term Memory Neural Networks. They collected data from accelerometer, gyroscope, and magnetometer sensors, dividing it into timestep windows of 100, 50, and 25. Initial experiments on a small drone (CrazyFlie 2.0) produced poor results, leading them to conduct further tests with a larger drone using Microsoft's AirSim simulator. These later experiments demonstrated good accuracy, as reported by the authors. However, a significant drawback is that the method was only tested in a simulated environment, raising concerns about its effectiveness in real-world scenarios.

Data-driven models, while powerful, have certain limitations. They often require large datasets for training and validation, and they can be sensitive to noise and outliers. While these models are valuable for modeling and evaluating complex systems, they must be used with caution, considering their limitations.

3 Background and Problem Definition

Drones are typically equipped with a variety of sensors capable of measuring different physical quantities and converting them into signals that can be interpreted by the onboard computer systems. These sensors are essential for monitoring the drone's status and its interaction with the environment.

To formalize this, let $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ represent the set of sensors deployed in a drone. In this work, we specifically experimented with the following sensors: accelerometer, magnetometer, gyroscope, barometer, and temperature sensors.

▶ **Definition 1** (Sensor). *Let $s_i \in \mathcal{S}$ denote a sensor within the drone's sensor set $\mathcal{S}$, s_i, reflects distinct observations or readings about the environment or the drone itself.*

These sensor readings are paramount, forming the foundational data points required for evaluating the reliability and functionality of the sensors, and consequently, ensuring the optimal performance of the drone. Each sensor provides a stream of data over time, which we can represent as an observation vector. This vector captures the sensor's readings within a specific time frame.

▶ **Definition 2** (Observation Vector). *For each sensor s_i, the collected values or readings over a period form an observation vector, $O^{s_i}_{j,j+t} = [o^{s_i}_j, o^{s_i}_{j+1}, \ldots, o^{s_i}_{j+t}]$, where each element represents an individual observation of sensor s_i from j to $j+t$.*

When considering all sensors together, we can aggregate their observation vectors over the same time frame into what we refer to as Window Observation Vectors. This gives a comprehensive snapshot of the drone's sensor readings during a particular interval.

▶ **Definition 3** (Window Observation Vectors). *We denote $O_{j,j+t}$ as the Window Observation Vectors, which consolidates all individual sensor observation vectors within a window time frame from j to $j+t$. Formally expressed as $O_{j,j+t} = \{O^{s_1}_{j,j+t}, O^{s_2}_{j,j+t}, \ldots, O^{s_n}_{j,j+t}\}$.*

Understanding the relationships between different sensors is crucial for fault detection. Sensors that are correlated may exhibit patterns that can indicate potential issues.

▶ **Definition 4** (Correlated Sensors). *Let $s_i, s_f \in \mathcal{S}$. The predicate $\sigma(s_i, s_f) = --xttrue$ asserts that sensors s_i and s_f are correlated, implying that their respective Observation Vectors exhibit a correlated relationship.*

In our implementation, we computed the correlation between sensors using Pearson correlation calculations, which identify meaningful relationships between sensor readings.

Observation Vectors can exhibit various patterns, where a pattern is defined as a specific trend or behavior in the sensor readings over a period of time. For instance, a "constant" pattern signifies that the sensor readings do not change over time. We categorize these patterns into a set $\mathcal{P}$, which includes patterns like "constant," "drift," and "abrupt."

▶ **Definition 5** (Pattern Recognition). *Let the function $\pi : O \to \mathcal{P}$ signify that a given Observation Vector $O^{s_i}_{j,j+t}$ exhibits a pattern $p \in \mathcal{P}$, where $\mathcal{P}$ is the set of all possible patterns.*

For example, the predicate $\pi(O^{s_i}_{j,j+t}) =$ "constant" if all values in the Observation Vector of sensor s_i within the time window j to $j+t$ are identical. Similarly, $\pi(O^{s_i}_{j,j+t}) =$ "drift" if the derivative of the values in the Observation Vector shows a consistent increase or decrease beyond a certain point in time. Finally, $\pi(O^{s_i}_{j,j+t}) =$ "abrupt" if the Observation Vector demonstrates a sudden and significant change in values, indicating a potential abrupt in the sensor readings.

In our research, we focus on detecting faults characterized by three primary patterns: abrupt, drift, and constant behaviors. These patterns often raise suspicion of malfunctions and can serve as indicators of potential sensor issues. In contrast, other patterns, such as fluctuations with varying intensities, are typically indicative of normal sensor operation. These normal patterns are explored further in the article in the evaluation section. Any sensor behavior can be considered a type of pattern, and while most are associated with normal drone function, recognizing and categorizing these patterns is essential for identifying potential malfunctions.

With these concepts in place, we can now define the Fault Detection Problem, which is central to our work. The goal is to determine whether any sensor within the drone is exhibiting patterns indicative of a fault.

▶ **Definition 6** (Fault Detection Problem). *Given the set of sensors $\mathcal{S}$ and their corresponding Window Observation Vectors $O_{j,j+t}$, the fault detection mechanism aims to determine whether sensor $s_i \in \mathcal{S}$ is exhibiting a pattern $x \in \mathcal{P}$ indicative of a fault.*

Accurately identifying correlated sensors, as represented by the predicate $\sigma(s_i, s_f)$ and recognizing predefined patterns within the sensor Observation Vectors, denoted by the predicate $\pi(O^{s_i}_{j,j+t}, p)$ are key components in approaches that rely on correlation-based fault detection. A robust solution must carefully analyze these correlations and patterns to differentiate genuine faults from innocuous variations or sensor noise, ensuring the reliability and safety of drone operations.

3.1 The sensor-based fault detection

Previous studies [11] proposed leveraging sensor correlations to identify faults in drone systems. The idea is to examine whether a sensor is displaying abnormal patterns, and then determine if this abnormality disrupts its usual correlation with other sensors. This brings us to the discussion of two significant methodologies:

Algorithm 1 Simple-SFDD.

Input: sensor data stream D, window size W, *threshold*
Output: faulty sensor identification F

```
for each sliding window do
    for each pair of sensors (i, j) do
        Calculate Pearson correlation coefficient ρ_{i,j} from first half of the window;
        if |ρ_{i,j}| > threshold then
            Mark sensors i and j as correlated;
        end
    end
end
for each sensor i in the second half do
    if i has a pattern of a fault then
        if no other correlated sensor in the first half of the window has the same pattern
         then
            Mark sensor i as faulty;
        end
    end
end
```

Simple-SFDD

The first strategy is named *Simple-SFDD* [12], and it operates in real-time. This method begins each flight with the assumption that all sensors are functioning correctly. It uses the initial portion of each sliding window to analyze the relationships between every pair of sensors, utilizing Pearson correlation to identify clear and linear relationships. A predetermined threshold is used to determine whether the sensors are indeed correlated. In the latter portion of the window, each sensor is assessed for abnormal patterns such as drift, abrupt, or constant readings. If a sensor exhibits such patterns, we look for another sensor that was correlated with it in the initial portion and is displaying the same abnormal pattern. If no such correlated sensor is found, the original sensor is flagged as faulty. This approach enables the timely detection of sensor faults, allowing for immediate corrective action and preventing potential damage. A detailed algorithm can be found in Algorithm 1.

The *Simple-SFDD* method analyzes correlations within a sliding window to identify temporal correlations. However, it has two key limitations. First, normally uncorrelated sensors may temporarily appear correlated, leading to false negatives. Second, sensors expected to be correlated may occasionally show a lack of correlation, resulting in false positives and incorrect fault reports. Additionally, the method's requirement for correlated sensors to exhibit identical patterns is overly strict. A more flexible approach that allows for varying patterns could improve the heuristic's versatility and applicability.

Extended-SFDD

The observed limitations of the Simple method paved the way for the development of an enhanced approach, referred to as *Extended-SFDD* [11] builds on its predecessor by integrating both online and offline components. The offline phase includes several key steps: First, feature extraction adds a virtual sensor s_d for each sensor s_i created from the differentials of s_i's raw readings. Next, the correlation detection process is conducted offline, using a fault-free historical record. This step evaluates every pair of sensors for correlation throughout the entire flight, ensuring that only correlations sustained over the full operation are considered, thus filtering out insignificant temporal discrepancies.

This approach allows for a more flexible fault detection heuristic by accommodating diverse patterns that represent normal behavior. For correlated sensor pairs, a tuple set is constructed offline, capturing the various patterns observed during normal flights. For example, if a sensor typically shows a 'drift', the correlated sensor's common patterns during normal operation – such as small fluctuations – are included in the set. These patterns are mapped to the corresponding dynamic window size, meaning that for a specific time window (e.g., 10 seconds), the system expects to see the 'drift' in one sensor and corresponding behavior, such as fluctuations, in the correlated sensor.

In the online phase, each time window during the flight is carefully analyzed. The focus is on identifying sensors that are now exhibiting patterns suggestive of a fault. If such patterns are detected, the system checks whether a correlated sensor is showing an unrecognized, abnormal pattern. For instance, if sensor X shows an abrupt pattern and a correlated sensor Y shows slight fluctuations, the system will verify whether this combination has been observed during normal flight conditions. If this combination of patterns has been seen before, the current operational state is considered normal, and no fault is flagged. However, if this combination is new or abnormal, it signifies a potential fault.

The *Extended-SFDD* method, while innovative, has its limitations. A key challenge is the requirement for sensors to maintain correlation throughout the entire flight, which can lead to the oversight of intermittently correlated sensors. While lowering the correlation threshold might address this, it risks falsely identifying uncorrelated sensors as correlated, which hinders accurate fault detection.

A more complex issue arises from the way sensors exhibit patterns in normal data. For example, a sensor displaying a drift pattern might normally correlate with another sensor that shows all possible patterns alongside the drift. As a result, if sensor X shows a drift pattern, it might never be flagged as a fault because there will always be a corresponding sensor Y exhibiting some pattern that has been observed before in normal conditions. This means that sensor X's drift will always appear normal, even if it indicates a fault, because the correlation with sensor Y's varied patterns has already been established as standard. This situation underscores the overly generalized nature of the fault detection heuristic, highlighting the need for refinement to accurately detect faults.

In light of these challenges, and the increased complexity within the domain of drones, we developed the methodologies presented in this article. Two new algorithms, *Combined-SFDD* and *Knowledge-Based-SFDD*, were created, both utilizing correlation detection. These methodologies will be discussed in the ensuing section.

4 Methodology

In this paper, we introduce two innovative methods for detecting faults in drones, with a focus on analyzing correlations within their operational components. These methods are crafted to address the shortcomings of previous correlation-based approaches, which have been found lacking in certain scenarios. By combining offline analysis with real-time monitoring, and leveraging a data-driven knowledge-based system, our methods offer a comprehensive and more effective solution. These approaches not only tackle the challenges faced by earlier models but also excel in situations where the original methods struggle due to the complex nature of data patterns and distributions.

4.1 Combined-SFDD

The first methodology, termed *Combined-SFDD* is a hybrid approach that integrates the strengths of both the Simple and Extended-SFDD methods. The goal is to leverage the advantages of each, leading to a more accurate and reliable fault detection system. This method operates in two phases: the offline phase and the online phase.

Offline Phase. The process begins with the execution of the Extended-SFDD method, where the following steps are performed offline:

1. For each identified feature, an additional feature is generated by calculating the differences between consecutive values of the original feature.
2. Sensors are correlated based on data from the entire flight duration, rather than within a temporary window, to identify sustained correlations over time.
3. For sensors that are correlated during normal flight conditions, a set of coexisting patterns is established and stored for reference.

Online Phase. In the online phase, flight data is continuously analyzed within a moving time window. Sensors are monitored for any patterns indicative of potential faults, such as constant, abrupt, or drift behaviors. If a sensor shows a suspicious pattern, the system first

checks whether a correlated sensor displays a pattern that has been previously identified as normal based on the offline analysis. If such a correlated sensor is found, the system considers the situation normal and no fault is flagged.

However, if no correlated sensor with a normal pattern is identified, the Combined-SFDD approach does not stop. It then applies the Simple-SFDD method to the same time window as an additional step. This involves examining the correlations observed in the first half of the time window and verifying whether these correlations persist in the second half. If the correlation fails to hold, a fault is declared.

This dual approach ensures that if the Extended-SFDD method fails to detect a fault, the Simple-SFDD method is used as a secondary check to identify issues that may have been missed. While this thorough process may slightly increase the likelihood of false positives, it significantly enhances the detection of true positives, ensuring that potential faults are not overlooked, especially in complex and varied operational scenarios.

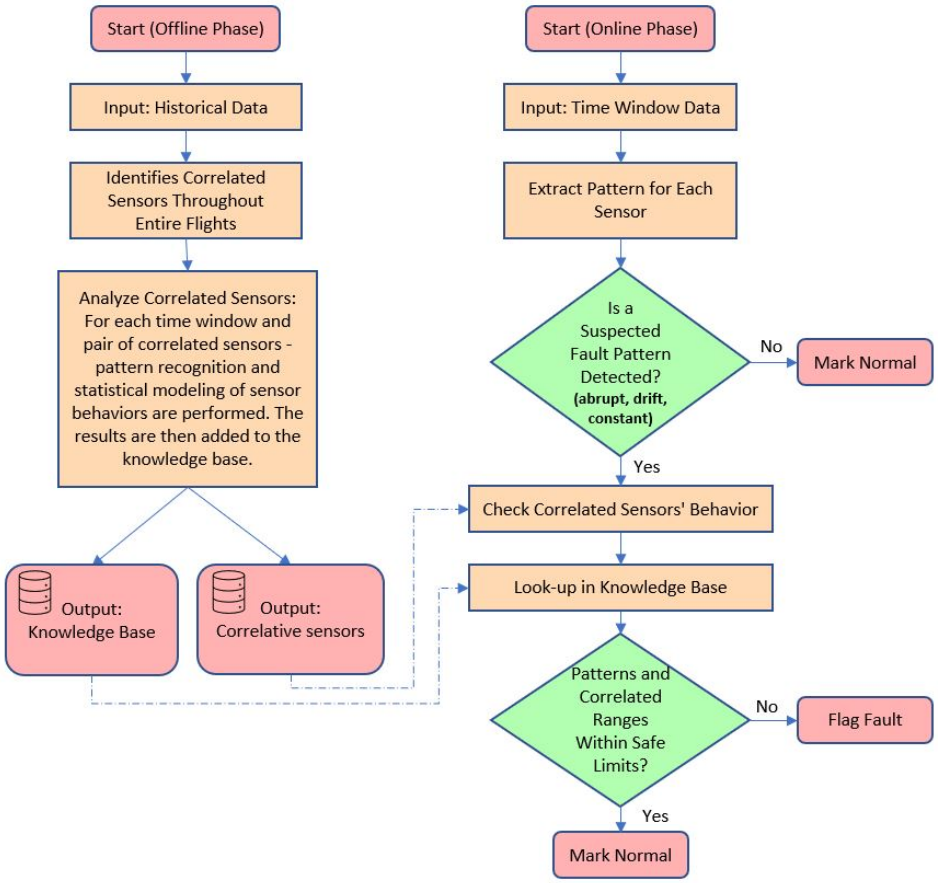


Figure 2 Flowchart of the Knowledge-Based-SFDD Algorithm. This diagram illustrates the process of offline and online phases for detecting sensor faults in real-time.

4.2 Knowledge-Based-SFDD

The *Knowledge-Based-SFDD* strategy introduces an advanced method that merges elements from Extended-SFDD with sophisticated knowledge-based system techniques. This approach is designed to deliver a deeper and more precise analysis of sensor behavior, providing a robust framework for fault detection.

Offline Phase. In the offline phase, Knowledge-Based-SFDD goes far beyond the basic steps outlined in the Combined-SFDD approach. This phase involves an exhaustive learning process that thoroughly analyzes all available data from previous flights, focusing on each sensor individually to understand its "normal" behavior and patterns. The aim is not merely to identify existing templates but to refine and model these templates within defined parameters.

For example, instead of simply noting that sensor X exhibits drifting while sensor Y shows slight fluctuations, Knowledge-Based-SFDD details the behavior more precisely: sensor X typically drifts at a slope between -5 and 5, while sensor Y fluctuates within a range of 2 to 4. By modeling the behavior of each sensor in this way, the system defines what constitutes normal and abnormal patterns, considering both the type of pattern and the specific range of values associated with it.

This meticulous modeling process allows for a comprehensive understanding of each sensor's behavior, distinguishing between normal variations and true anomalies. All identified patterns and correlations, along with their precise parameters, are compiled into an extensive knowledge base. This repository serves as a reference for real-time analysis, enabling the system to distinguish between normal operational variations and genuine anomalies effectively.

Online Phase. In the online phase, Knowledge-Based-SFDD continuously monitors real-time flight data within a moving time window, meticulously analyzing sensor readings for patterns that might indicate potential faults, such as constant values, abrupt changes, or unusual drifts. When a suspicious pattern is detected, the system references the comprehensive knowledge base developed during the offline phase to determine whether a correlated sensor exhibits a corresponding normal pattern within the expected parameter ranges. This assessment is not limited to merely recognizing patterns; it involves a detailed evaluation of whether the detected patterns fall within the predefined safe operational limits. If a correlated sensor is found that displays a recognized and acceptable pattern within these ranges, the system considers the situation normal, and no fault is declared. However, if no such correlated sensor is identified, or if the patterns observed fall outside the established safe ranges, the system conclusively flags the sensor as faulty. This rigorous process ensures a more accurate and reliable fault detection, minimizing false positives while effectively identifying genuine anomalies.

In summary, the process of Knowledge-Based-SFDD, including its offline and online phases, is visually represented in Figure 2. This flowchart provides a clear depiction of how the algorithm models sensor behavior, checks correlated patterns, and makes real-time decisions based on predefined parameters.

Benefits of Knowledge-Based-SFDD: The precision of Knowledge-Based-SFDD offers several critical advantages:

- **Enhanced Accuracy:** By defining normal behavior with specific parameters, Knowledge-Based-SFDD significantly reduces the risk of false positives and negatives, ensuring that only genuine faults are flagged.
- **Comprehensive Analysis:** This approach provides a holistic understanding of sensor interactions, considering both the patterns and their permissible ranges, which leads to more informed and accurate fault detection.
- **Adaptability:** The model adapts to the unique characteristics of each sensor, offering a tailored analysis that can accommodate various operational scenarios and sensor configurations.

By modeling the entirety of each sensor's behavior, Knowledge-Based-SFDD offers a nuanced and highly accurate fault detection system. This method not only identifies deviations from expected patterns but also assesses the severity and relevance of these deviations in real-time. The result is a robust and reliable system that enhances the safety and efficiency of drone operations, ensuring that anomalies are detected early and accurately.

5 Evaluation

Next, we precisely define the experiments we conducted to validate our methodologies. This will include detailed descriptions of the test scenarios, the datasets used, and the specific metrics employed to assess performance. Following that, in the Results chapter, we will present and analyze the outcomes of these experiments, demonstrating the effectiveness and superiority of our proposed methods, Combined-SFDD and Knowledge-Based-SFDD, in accurately detecting faults under various conditions.

5.1 Data set

One of the primary contributions of this study is the use of real-world data, collected in real-time during the communication between a drone and its control unit. This data is subject to various external factors such as weather conditions, heat, and humidity, which can significantly influence the drone's behavior and sensor readings. We have taken meticulous care to ensure the accuracy and reliability of the data used in our experiments, thereby guaranteeing that the results obtained are robust and applicable to real-world scenarios.

Collecting such data presents significant challenges. To overcome these obstacles, we partnered with Maris, a company specializing in Intelligent Video Surveillance and Analytics. Maris provided us with the technical expertise and tools necessary for capturing high-quality, real-world data.

For our study, we used drones from Aero Sentinel, a leading manufacturer in the tactical UAV industry, known for its high-performance military drones used in various applications, including combat, Homeland Security (HLS), police operations, and civilian use. We leveraged a substantial amount of raw flight data collected from these actual Aero Sentinel drones. This collaboration allowed us to gather a comprehensive dataset that includes detailed sensor data recorded during real-world drone operations under various environmental conditions.

The data was gathered during controlled flights, with sensor data recorded and stored in files, each containing approximately 35 minutes of flight time. A total of 30 flights were conducted, resulting in about 17 hours of documented flight data. To maintain consistency, all flights were conducted using the same drone model: Aero Sentinel Military UAV Sentinel G2 [4]). This extensive dataset comprises approximately 611,000 records, underscoring the substantial volume of data collected for analysis and model training.

The controlled flights involved testing a variety of real-world flight scenarios to capture diverse operational conditions. These included hovering at different altitudes under varying wind speeds, flights at multiple velocities, and takeoff and landing maneuvers at different angles. By incorporating such a range of flight conditions, we were able to simulate the types of environments a drone might encounter in practice, ensuring that the dataset reflected a comprehensive spectrum of sensor behaviors.

[4] https://www.aero-sentinel.com/military-drones/military_drone_sentinel_g2/

Table 1 Summary information for the sensors.

Sensor	Measure	Description	Units
Accelerometer	Xacc	X acceleration	int
	Yacc	Y acceleration	int
	Zacc	Z acceleration	int
Magnetometer	Xmag	X Magnetic field	int
	Ymag	Y Magnetic field	int
	Zmag	Z Magnetic field	int
Gyroscope	Xgyro	Angular speed around X axis	int
	Ygyro	Angular speed around Y axis	int
	Zgyro	Angular speed around Z axis	int
Barometer	Press abs	Absolute pressure	hPa
	Temperature	Absolute pressure temperature	cdegC

The collected data encompasses readings from all of the drone's sensors, recorded at varying sampling intervals for each sensor. Our study specifically focused on sensors providing information about gyroscope, accelerometer, magnetometer, barometer, and drone temperature. Detailed information about the available sensors is provided in Table 1.

The dataset we created, with the help of capturing and translating packets transmitted over the network during real flights, represents a significant contribution of this study. Developing this dataset was a challenging process, involving the meticulous collection and interpretation of data under diverse environmental conditions. After extensive pre-processing operations, this unique dataset is now available for use, offering a rich and authentic resource that is crucial for evaluating and improving fault detection methods in drones. Its real-world applicability makes it an invaluable tool for advancing research and ensuring that our findings are relevant and effective in practical scenarios.

The entire dataset, including the original data and the data post-fault injection, is available in the parent Git repository. Access is via the following link:
`https://github.com/inbalros/SensorFaultDetection_Correlation`

Data Collection and Processing

Sensor data was collected from the communication between the drone and its control unit, recorded in a binary LOG file. Using the Mavlink v1 protocol, relevant sensor data packets were identified and processed, with timestamps provided every 0.1 seconds to create a time-series dataset for fault prediction.

Due to the high data collection frequency, a large volume of records was generated, requiring significant processing time. To optimize this process, we explored whether resampling the data at longer intervals, specifically half a second and one second, could still maintain accuracy while reducing the dataset size. Although this approach decreased the number of records, initial experiments showed that the algorithm's performance was less effective compared to using the full dataset at the 0.1-second frequency.

After collecting the data, we encountered two challenges in creating a usable tabular dataset. The first challenge involved handling multiple samples recorded from certain sensors within the same time interval. The second issue was dealing with missing samples from other sensors during certain time intervals.

To solve the multiple samples problem, we implemented two methods:

1. Avg_{comb} **(Averaging Combination):** This method averages the values when multiple samples exist within a time window and fills in missing values accordingly.
2. $Last_{comb}$ **(Last Value Combination):** This method takes the last value in cases of multiple samples within a time window and completes any missing values.

Preliminary experiments indicated that the $Last_{comb}$ method yielded the best results.

To address the missing values issue, four strategies were employed for imputing the missing data:

1. $Last_{missing}$ **(Last Value Imputation):** This method fills in missing values with the last available value for that sensor. If no prior value exists, a default value is assigned.
2. $SMA_{missing}$ **(Simple Moving Average):** A weighted moving average is calculated using the last 10 samples from the same sensor, with statistical prediction applied to estimate the missing value. If fewer than 10 samples are available, a default value is used.
3. $LR_{missing}$ **(Linear Regression):** Linear regression is applied by gathering all available samples of the sensor up to the missing point and predicting the missing value based on this data.
4. $ARIMA_{missing}$ **(Autoregressive Integrated Moving Average):** ARIMA, a time series forecasting model, is used to predict missing values. This model requires careful parameter configuration, achieved through an offline grid search to identify the best-performing ARIMA model for each sensor. The chosen ARIMA model is then used to predict missing values based on all available sensor data up to that point.

Extensive experiments demonstrated that the $Last_{missing}$ method consistently provided the best results in terms of accuracy and reliability.

Simulating Sensor Faults in Flight Data

A critical challenge in this research is the lack of real fault data, as the available dataset only contains normal flight information. To address this, we employed simulation techniques to generate artificial sensor faults, which were then integrated into the normal flight data. This approach allows for effective training and evaluation of our fault detection models.

An automated simulation tool was developed to generate data files containing simulated faults, providing a controlled environment for testing and refining our models. The types of faults simulated include: (1) *Constant Faults:* Represent a persistent deviation from normal sensor readings. (2) *Drift Faults:* Describe a gradual change in sensor readings over time. (3) *Abrupt Faults:* Indicate sudden and significant changes in sensor readings, which can be expressed as sharp increases or decreases.

To ensure realistic simulations, expert insights from the Maris and Aero-Sentinel companies were combined with thorough statistical analysis of the sensor data. This analysis helped establish the range of values each sensor can sample, including averages and standard deviations, which served as guidelines for creating realistic fault simulations. The simulated faults were carefully designed to stay within the possible value ranges for each sensor.

The following ranges were defined for each type of fault:

1. constant:
   - Fixed to the first instance value within the current time window.
   - Fixed to the maximum possible sensor value.
   - Fixed to the minimum possible sensor value.
   - Fixed to 0, simulating a scenario where the sensor stops receiving information (if 0 is a possible value for that sensor).

2. drift:
   - A decrease from the first value in the time window, following the steepest negative slope observed for that sensor.
   - An increase from the first value in the time window, following the steepest positive slope observed for that sensor.
3. abrupt:
   - A sudden increase from the initial value in the time window, based on the largest positive jump the sensor is capable of.
   - A sudden decrease from the initial value in the time window, based on the largest negative drop the sensor is capable of.

To ensure comprehensive testing, the normal flight data was segmented into slightly larger time windows than those used in the fault detection algorithm. Three distinct data files were created, each containing a different type of fault: constant, abrupt, and drift. Faults were systematically introduced into every second time window, covering all sensors and representing all possible values of the specific fault type for each file. In total, 90 files (3 types of faults $\times$ 30 different scenarios) were generated, each containing various faults across different sensors and flight conditions.

5.2 Competing Approaches and Pattern Analysis

To evaluate the effectiveness of our correlation-based algorithm, several different approaches were tested:

1. **Simple-SFDD:** This method, based on the work of Khalastchi et al. [11], uses a straightforward approach to detect faults by analyzing sensor correlations within a dynamic time window. Further details can be found in the Background section (3.1).
2. **Extended-SFDD:** An enhancement of the Simple-SFDD, based on the work of Khalastchi et al. [11], this method introduces offline analysis to refine correlation detection, making it more robust. It is also detailed in the Background section (3.1).
3. **Combined-SFDD:** Building on the principles of Simple and Extended-SFDD, this method integrates both approaches to improve fault detection accuracy and is specifically adapted for drone operations. This method is discussed in detail in the Methodology section (4).
4. **Knowledge-Based-SFDD:** An advanced method that incorporates knowledge-based system techniques to model and analyze sensor behavior, offering enhanced fault detection capabilities. This approach is also detailed in the Methodology section (4).

5.3 Experimental Process and Setup

To evaluate the effectiveness of our fault prediction model, we conducted a series of experiments using K-fold cross-validation, with K set to 3. In each experiment, 20 flight files were randomly selected for training, while 10 flight files were designated for testing. These test files were drawn from the simulated fault files generated during the fault simulation phase, with each test file corresponding to three different simulated fault scenarios, resulting in a total of 30 variations of test files.

The training data plays a crucial role in the offline phases of the correlation-based algorithms, allowing the models to learn and adapt. In contrast, the testing data, used for the online phase only, (the Simple-SFDD approach, which operates solely online) did not utilize the training files within its fold.

For correlation-based detection, we applied a Pearson correlation coefficient of 40% between two sensors to identify meaningful relationships and detect faults.

Additionally, during the Knowledge-Based-SFDD algorithm, we managed the Knowledge Base to contain the modeling of patterns that were suspected to indicate faults, such as abrupt and drift patterns. The Knowledge-Based approach not only identifies patterns but also models their specific values, such as the magnitude of the jump in abrupt patterns and the slope in drift patterns, against the maximum possible values observed for two correlated sensors. This modeling is context-specific. Instead of comparing a sensor's behavior across its entire history, we analyze how a pattern like a drift behaves when a correlated sensor exhibits a particular pattern.

Moreover, we tested different sizes of time windows across all four algorithms: 6 seconds, 10 seconds, 14 seconds, and 20 seconds. The number of instances varied depending on the data representation method used. For raw data, where each record represents 0.1 seconds, a 10-second window contains 100 records.

5.4 Evaluation Metrics

In addressing the problem outlined in this article, the most relevant metrics for evaluating the success of our fault detection algorithms are the True Positive Rate (TPR) and the False Positive Rate (FPR).

True Positive Rate (TPR). This metric measures the proportion of time windows where the algorithm correctly identified a fault when one was actually present. It is calculated as the number of true positives (correctly identified faults) divided by the total number of actual faults ($TP/(TP + FN)$). Importantly, a true positive (TP) is only considered valid if the fault is detected in the correct sensor. If the algorithm detects a fault but incorrectly diagnoses it in a different sensor, this is not counted as a correct detection. Thus, TPR reflects the algorithm's success rate in accurately detecting both the fault and the affected sensor. Essentially, TPR reflects the algorithm's success rate in detecting genuine faults.

False Positive Rate (FPR). This metric indicates the proportion of time windows where the algorithm incorrectly flagged a fault when none existed. It is calculated as the number of false positives (incorrectly identified faults) divided by the total number of windows where no fault was present ($FP/(FP + TN)$). In addition, FPR includes cases where a fault is present, but the algorithm incorrectly identifies the wrong sensor. So, even if a fault exists, if the diagnosis points to the wrong sensor, this is counted as a false positive, highlighting the algorithm's error rate in both detecting non-existent faults and misdiagnosing sensor faults.

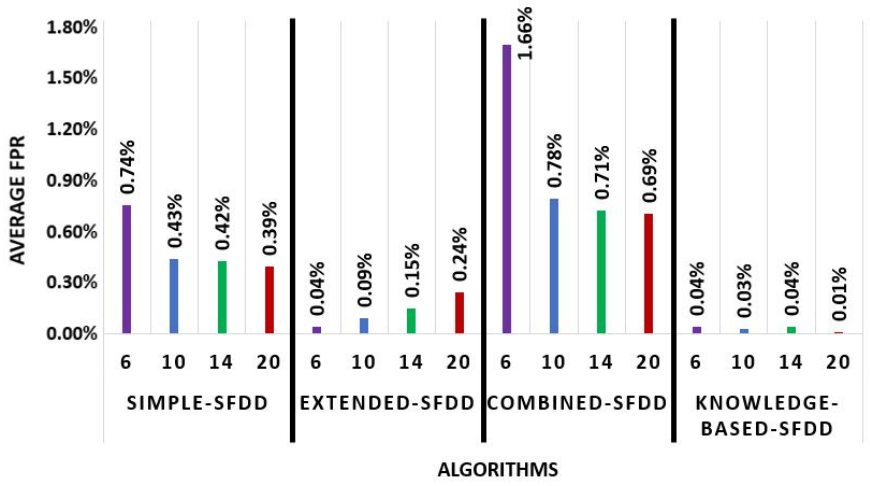
In addition to these primary metrics, we also considered using Precision and Recall. Recall, is identical to TPR, measuring the accuracy of the algorithm in identifying actual faults. Precision, evaluates how many of the time windows identified as faulty by the algorithm were indeed faulty, out of all the time windows flagged as faulty, whether correctly or not. However, Precision may be less relevant in our context due to the significant imbalance between the number of time windows without faults and those with faults.

6 Results

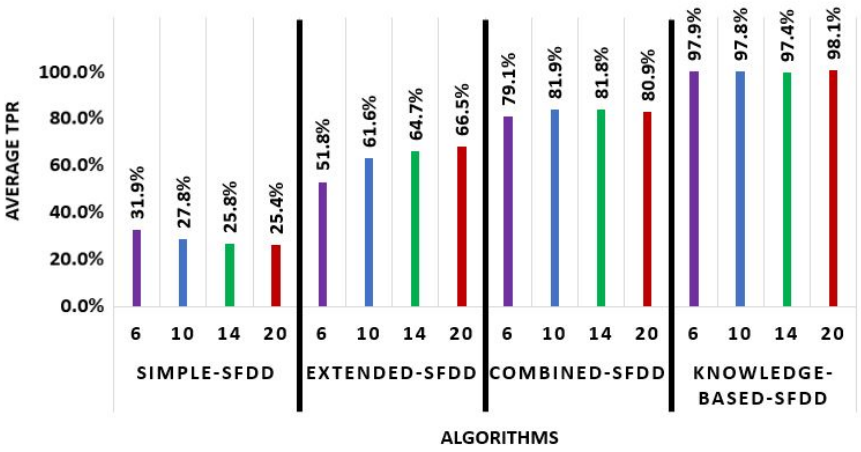
In the following figures 3a, 3b, we present a comparison of the four algorithms – Simple-SFDD, Extended-SFDD, Combined-SFDD, and Knowledge-Based SFDD – using two key metrics: Average False Positive Rate (FPR) and Average True Positive Rate (TPR). These graphs visually demonstrate how each algorithm performs across various time window sizes (6, 10, 14, and 20 seconds), providing a clearer illustration of the algorithm's effectiveness.

Figure 3a depicts the Average FPR for each algorithm. A lower FPR indicates fewer instances of false fault detection, where a malfunction is falsely identified. It can be observed that Knowledge-Based SFDD consistently outperforms other methods, achieving the lowest FPR across all window sizes, while Combined-SFDD demonstrates the highest FPR values.

Figure 3b shows the Average TPR, which reflects the accuracy of each algorithm in detecting genuine faults. A higher TPR suggests better fault detection. Once again, Knowledge-Based SFDD stands out with the highest TPR in all configurations, while Simple-SFDD yields the lowest TPR.



(a) Average False Positive Rate (FPR).



(b) Average True Positive Rate (TPR).

Figure 3 Comparison of FPR and TPR for Simple-SFDD, Extended-SFDD, Combined-SFDD, and Knowledge-Based SFDD across different time windows (6, 10, 14, and 20 seconds).

The results show that our Knowledge-Based-SFDD algorithm consistently delivers excellent performance, achieving an average TPR of 98%, with an impressively low FPR of just 0.03%. This indicates that the Knowledge-Based-SFDD not only accurately predicts malfunctions but also minimizes the instances of false alarms – where a fault is indicated, but none exists. The model's algorithm systematically outperforms the other approaches across all configurations, clearly demonstrating its robustness and reliability.

The Simple-SFDD, which operates exclusively online, achieves the lowest success rates in fault detection. Its performance is hampered by its reliance on short-term correlations, which often fail to capture the complexity of sensor interactions over time. Consequently, it struggles with both lower TPR and a higher incidence of false positives. Expanding the window size does little to alleviate these issues, as the difficulty in detecting correlations within larger windows further complicates fault detection.

The Extended-SFDD improves upon the Simple approach by incorporating offline analysis, yet it still falls short in several areas. The primary limitation here is that the presence of multiple potential patterns for each sensor makes it difficult to pinpoint specific faults. However, as the window size increases, the algorithm gains confidence in identifying patterns, resulting in a gradual improvement in TPR.

The Combined-SFDD, which integrates elements of both the Simple and Extended approaches, shows a notable improvement in fault detection. In this method, if the Extended-SFDD does not detect a fault, the algorithm applies the Simple-SFDD, but with additional features such as differentials. This enhanced version of the Simple-SFDD allows it to detect more faults than the standard Simple approach, as it now takes into account the extra features. However, this comes at the cost of a higher FPR, indicating that while the Combined-SFDD is more sensitive to faults, it is also more prone to false positives due to the broader feature set being examined.

The Knowledge-Based-SFDD emerges as the most effective algorithm across all configurations. It consistently delivers the highest TPR and the lowest FPR, making it the best choice for reliable fault detection. The model's ability to learn and model complex sensor behaviors, combined with the increased confidence provided by larger window sizes, ensures that it excels in both detecting true faults and avoiding false alarms.

7 Conclusions and Future Work

Drones, or unmanned aerial vehicles (UAVs), play an increasingly important role across various industries, making their reliability and safety-critical. A major challenge in maintaining drone operations is the detection and diagnosis of sensor faults, which can significantly impact performance. In this study, we focused on detecting these sensor-related faults to enhance the safety and efficiency of drones. Through a crucial collaboration with Maris, a pioneer in AI edge video and analytics technologies, we gathered a comprehensive dataset from real drones. Utilizing Maris's advanced Mercury Nano system, we recorded the communication between an Aero-Sentinel Military UAV Sentinel G2 quadcopter drone and its control unit, capturing detailed sensor data during operation. This partnership was instrumental in providing the high-quality data that formed the foundation of our research. We developed and tested correlation-based algorithms, including two new methods – Combined-SFDD and Knowledge-Based-SFDD specifically designed to address the complexities of drone sensor data. Our experiments demonstrated the clear superiority of these algorithms, particularly the Knowledge-Based-SFDD, which achieved a 98% True Positive Rate (TPR) and a low False Positive Rate (FPR) of 0.03%. These results highlight the effectiveness of our approach in accurately detecting faults while minimizing false alarms.

Looking forward, several avenues for enhancing and expanding this research can be explored. One promising direction is the integration of audio data into the fault detection process. By incorporating sound analysis, we can potentially identify and diagnose faults that may not be detectable through sensor data alone, further improving the accuracy and reliability of the detection system. Another key area for future work is the exploration and

comparison of deep learning algorithms for detecting structural faults in drones. While our current study focused on sensor-related faults, structural faults – such as those affecting the drone's frame or mechanical components – pose significant risks to drone operations. Leveraging deep learning techniques could provide more advanced and effective methods for identifying these types of faults, thereby enhancing the overall safety and performance of drone systems.

References

1 Mogens Blanke and Søren Hansen. Towards self-tuning residual generators for uav control surface fault diagnosis. In *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 37–42. IEEE, 2013.

2 Mark Bowkett, Kary Thanapalan, and Ewen Constant. Failure detection of composites with control system corrective response in drone system applications. *Computers*, 7(2):23, 2018. doi:10.3390/COMPUTERS7020023.

3 Mingzhang Chen, Xuancheng Zhang, Xiaoshuang Xiong, Fanfei Zeng, and Wuhao Zhuang. Transformer: A multifunctional fast unmanned aerial vehicles–unmanned surface vehicles coupling system. *Machines*, 9(8):146, 2021.

4 Yuepeng Chen, Xia Hu, Qingyong Zhang, and Cong Zhang. Research on multi-classification method of uav sensor fault based on wavelet entropy and afwa-bp neural network. In *2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 837–842. IEEE, 2017.

5 Yuepeng Chen, Cong Zhang, Qingyong Zhang, and Xia Hu. Uav fault detection based on ga-bp neural network. In *2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 806–811. IEEE, 2017.

6 Merve Demircan and Coşku Kasnakoğlu. Aileron locking fault detection based on extended kalman filter for uav. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, pages 1–6, 2019.

7 Xingjian Fu and Xinyao Geng. Fault estimation and robust fault-tolerant control for singular markov switching systems with mixed time-delays and uav applications. *Journal of Control Engineering and Applied Informatics*, 23(2):53–66, 2021.

8 Hussein Hamadi, Benjamin Lussier, Isabelle Fantoni, and Clovis Francis. Data fusion fault tolerant strategy for a quadrotor uav under sensors and software faults. *ISA transactions*, 2022.

9 Samiadji Herdjunanto. Actuator fault signal isolation of an unmanned aerial vehicle (uav) quadrotor based on detection filter. In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–6. IEEE, 2016.

10 Gino Iannace, Giuseppe Ciaburro, and Amelia Trematerra. Fault diagnosis for uav blades using artificial neural network. *Robotics*, 8(3):59, 2019. doi:10.3390/ROBOTICS8030059.

11 Eliahu Khalastchi and Meir Kalech. A sensor-based approach for fault detection and diagnosis for robotic systems. *Autonomous Robots*, 42:1231–1248, 2018. doi:10.1007/S10514-017-9688-Z.

12 Eliahu Khalastchi, Meir Kalech, and Lior Rokach. Sensor fault detection and diagnosis for autonomous systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 15–22, 2013. URL: http://dl.acm.org/citation.cfm?id=2484927.

13 Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. Kalman filter and its application. In *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 74–77. IEEE, 2015.

14 Hamid Maqsood, Muhammad Taimoor, Zahid Ullah, Nihad Ali, and Muhammad Sohail. Novel sensor fault detection and isolation for an unmanned aerial vehicle. In *2021 International*

Bhurban Conference on Applied Sciences and Technologies (IBCAST), pages 486–493. IEEE, 2021.

15 Francisco J Montáns, Francisco Chinesta, Rafael Gómez-Bombarelli, and J Nathan Kutz. Data-driven modeling and learning in science and engineering. *Comptes Rendus Mécanique*, 347(11):845–855, 2019.

16 Radosław Puchalski and Wojciech Giernacki. Uav fault detection methods, state-of-the-art. *Drones*, 6(11):330, 2022.

17 Vidyasagar Sadhu, Saman Zonouz, and Dario Pompili. On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification. In *2020 ieee international conference on robotics and automation (icra)*, pages 5255–5261. IEEE, 2020. doi:10.1109/ICRA40945.2020.9197071.

18 Hazim Shakhatreh, Ahmad H Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *Ieee Access*, 7:48572–48634, 2019. doi:10.1109/ACCESS.2019.2909530.

19 Hassan Shraim, Ali Awada, and Rafic Youness. A survey on quadrotors: Configurations, modeling and identification, control, collision avoidance, fault diagnosis and tolerant control. *IEEE Aerospace and Electronic Systems Magazine*, 33(7):14–33, 2018.

20 Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N Kavuri. A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. *Computers & chemical engineering*, 27(3):293–311, 2003. doi:10.1016/S0098-1354(02)00160-6.

21 Benkuan Wang, Datong Liu, Yu Peng, and Xiyuan Peng. Multivariate regression-based fault detection and recovery of uav flight data. *IEEE Transactions on Instrumentation and Measurement*, 69(6):3527–3537, 2019. doi:10.1109/TIM.2019.2935576.

22 Yujiang Zhong, Wei Zhang, and Youmin Zhang. Sensor fault diagnosis for unmanned quadrotor helicopter via adaptive two-stage extended kalman filter. In *2017 international conference on sensing, diagnostics, prognostics, and control (SDPC)*, pages 493–498. IEEE, 2017.

23 Yujiang Zhong, Youmin Zhang, Wei Zhang, Junyi Zuo, and Hao Zhan. Robust actuator fault detection and diagnosis for a quadrotor uav with external disturbances. *IEEE Access*, 6:48169–48180, 2018. doi:10.1109/ACCESS.2018.2867574.

Simulation-Based Diagnosis for Cyber-Physical Systems - A General Approach and Case Study on a Dual Three-Phase E-Machine


David Kaufmann
CD Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems, Institute of Software Technology, Graz University of Technology, Austria


Matus Kozovsky
Central European Institute of Technology, Brno University of Technology, Czech Republic


Franz Wotawa
CD Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems, Institute of Software Technology, Graz University of Technology, Austria

Abstract

This paper presents a simulation-based approach for fault diagnosis in cyber-physical systems. We utilize simulation models to generate training data for machine learning classifiers to detect faults and identify the root cause. The presented processing pipeline includes simulation model validation, training data generation, data preprocessing, and the implementation of a diagnosis method. A case study with a dual three-phase e-machine highlights the results and challenges of the simulation-based diagnosis approach. The e-machine simulation model provides a complex and robust system representation, including the capability to inject inter-turn short-circuit faults. The introduced validation procedures of the simulation model revealed limitations in signal similarity and distinguishability compared to real system behavior. Based on the discovered limitations, the overall best results are achieved by applying an Autoencoder model for anomaly detection, followed by a Random Forest classifier to identify the specific anomalies. Further, the focus is on identifying the affected e-machine phase rather than the exact number of faulty winding turns. The paper shows the challenges when applying a simulation-based diagnosis approach to time-series data and underlines the required analysis of simulation models. In addition, the flexible adaption in the diagnosis strategies enhances the efficient utilization of cyber-physical system models in fault diagnosis and root cause identification.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Cyber-Physical System, Fault diagnosis, Root cause analysis, Simulation-Based Diagnosis, Machine Learning, Artificial Neural Networks

Digital Object Identifier 10.4230/OASIcs.DX.2024.18

Supplementary Material *Software (Method and Source Code)*: `https://doi.org/10.5281/zenodo.14026570` [7]

Funding *David Kaufmann*: The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged.
Matus Kozovsky: The work has been performed in the project A-IQ Ready: Artificial Intelligence using Quantum measured Information for realtime distributed systems at the edge No 101096658/9A22002. The work was co-funded by grants of Ministry of Education, Youth and Sports of the Czech Republic and Chips Joint Undertaking. The work was supported by the infrastructure of RICAIP that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306 and from Ministry of Education, Youth and Sports under OP RDE grant agreement No CZ.02.1.01/0.0/0.0/17_043/0010085.
Franz Wotawa: The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged.



© David Kaufmann, Matus Kozovsky, and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The automotive industry is experiencing a deliberate and long-planned transformation towards electrified vehicles, primarily affecting the powertrain and safety assistance systems. When vehicles are operated on the road, unsteady conditions and challenging environments may be encountered. For instance, extreme temperatures, changing transportation loads, high-speed driving, stop-and-go traffic, or even off-road driving. These scenarios can negatively impact the lifetime of a powertrain system and even cause unexpected failures. To enhance the reliability and safety of powertrain systems, it is important to address the component design and the specification of the operational design domain during the early development phase. In this context, fail-safe and fail-operational powertrain systems are mandatory to improve reliability, safety, and performance. With a focus on electric machine components of powertrains, research shows that most faults are related to issues with stator windings, bearing faults, magnet demagnetization, unbalanced magnetic pull, and eccentricity faults [3]. To give an example, a promising architecture for achieving a fail-operational design is the dual three-phase e-machine [13]. This architecture can operate even under stator fault conditions by appropriately adapting settings of power inverters, motor parameters, and control algorithms [12]. However, stator winding faults, like inter-turn short circuits (ITSC), can significantly impact motor performance, causing large currents and demagnetizing magnets [2] and therefore require a fast identification and mitigation. Thus, it is essential to identify the root cause of the failure in the system to take appropriate mitigation actions or schedule maintenance before a critical scenario occurs. To address this challenge, we propose a procedure using simulation models as virtual laboratories for generating normal and fault-injected data to train robust machine learning (ML) algorithms for early anomaly detection and root cause identification on real cyber-physical systems (CPS). For instance, an abstracted conceptual framework of the procedure is presented in [23], focusing on detecting bearing faults in wind turbine generators. Applying models for diagnosis in the classical model-based diagnosis [20] is a powerful approach that enables reasoning about the system behavior for systematically identifying faults or anomalies within complex systems. The model-based diagnosis approach avoids the expensive part of simulation [25] by an abstract representation of the system's properties and conditions. A successful demonstration of fault detection and root cause analysis by utilizing a model-based diagnosis on a CPS is presented by Wotawa et al. [8]. However, formalizing the logical representation of a CPS could be complex and challenging for dynamic behavior and a wide-ranged system's operational domain. A simulation-based approach could avoid the drawbacks since a deep knowledge of the model and related properties is not required. Nevertheless, potential constraints are also obvious because the simulated representation of real system behavior has to be as similar as possible to guarantee a precise fault diagnosis solution. In this paper, we address this challenge in a case study by training fault detection methods on simulated data to be tested on real measured data of an electrified machine to assess how ML methods manage deviations and gaps in the data. We introduce a comprehensive general approach encompassing the initial model specification, an extended model validation methodology aimed at gathering comprehensive information regarding potential applications, using the model to generate simulation data, preprocessing techniques for the collected data, and finally, the training and evaluation of ML-based fault classifier algorithms. Further, we take advantage of elaborated insights and present a case study with a complex CPS, a dual three-phase e-motor model capable of simulating a variation of stator issues in the context of short-circuit faults. With this case study, we demonstrate the identification of possible limitations of simulation models and introduce a potential fault detection method operated on real measured data.

2 General Approach

In this section, we present an overview of utilizing simulation models to produce training data for ML algorithms aimed at identifying unexpected behavior (faults) within the operational domain of a CPS. The authors in [23] followed a similar approach to perform condition monitoring and predict faults in rotating machine bearings based on relatively simple simulation models validated on experimental data of a real system representation. Besides statistical feature-based ML classifiers, convolutional neural networks were applied and evaluated based on their performance in fault prediction. The authors claimed that a purely simulation-data-driven approach cannot replace the experimental data. Kozovsky et al. [14] evaluated another approach, utilizing an experimental hardware testbench of a dual three-phase e-motor capable of injecting inter-turn short-circuit (ITSC) faults. The setup was used to collect signal data on normal and faulty system behavior as a basis for ML and Artificial Neural Network (ANN) classifier models for detecting and identifying ITSC faults during system operation. The validation on different speed and load profiles showed 99.8 % accuracy in detecting faults for the critical high-speed range above 800 rpm. A constraint of the methods employed is that specific critical faults cannot be entirely tested on a real test bench due to possible expensive system degradation or damage, leading to insufficient data representation for these faults. Utilizing digital models within a simulated environment can address the absence of data by generating synthetic training data for scenarios that are not covered, such as fault-injected cases. In this paper, we follow the idea and utilize a simulation model [13] of an electrical machine as discussed in [14] to train ML and ANN classifiers on simulated data and evaluate the fault detection rate and time efficiency when applied to real system measurements of the experimental testbench. The objective is to evaluate the potential of utilizing generated training data generated with the simulation model to enhance the fault detection accuracy of ML classifier models.

In [10], we present a basic methodological framework focusing on the processing pipeline and its application. Figure 1 extends this processing pipeline to support more comprehensive applications. The process starts with validating a CPS simulation model, using the gathered information for the data generation, and applying preprocessing methods such as feature selection and data extraction. Next, the data is used to train ML classifiers to detect system faults during runtime. The utilized example in [10] was a simple DC motor capable of injecting battery, torque, and motor faults. Due to the lack of real measurements, only the concept was evaluated. With this paper, we make use of the framework presented in Figure 1 and assess the approach to a complex CPS with available real experimental measurements of normal and faulty system conditions. It is important to mention that the algorithms and methodologies outlined in the paper were specifically designed and tested to handle time-series data gathered from respective CPSs. Further, we enhance the framework discussed with the CPS models' validation procedure to determine the similarity and distinguishability between the simulation and real measured data. This process supports the identification of constraints regarding the applicable operational domain of the CPS model.

The following description starts with an introduction to model generation, validation, execution to generate training data, the preprocessing mechanism, possible state-of-the-art classifier models, and an application with evaluation. The following definitions apply to the overall description:

- **Simulated/Simulation Data** - This refers to data generated by a CPS simulation model within a simulation environment. It represents real-world scenarios but is created artificially within a controlled virtual framework.

- **Measured/Measurement Data** - This term denotes data acquired from real hardware systems under laboratory conditions. It involves the direct recording of observations from physical components and applied software.

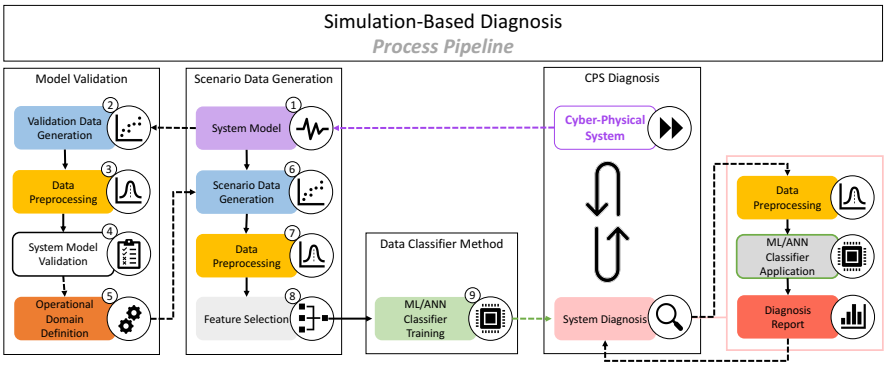


Figure 1 Simulation-based Diagnosis general pipeline process.

2.1 CPS Model Specification

CPSs combine physical and computational processes. The physical components, like sensors, actuators, and machinery, interact with computational elements, such as software, algorithms, and communication networks, to bridge the gap between the physical and digital worlds. CPSs are very prominent in different domains, including autonomous driving systems. In this paper, we focus on a main component in the automobile sector, the e-motor (physical) with speed controller algorithms (digital) and the application of fault detection methods (digital). To develop and test new algorithms and methods for such systems, it is obvious that a real physical system is not always applicable in terms of fast, reliable, cost-efficient, and safe execution. Thus, representative simulation models are introduced, modeling the physical component to be executed in a simulated environment. Although that approach comes with limitations, in terms of covered operational domain and realistic behavior representation, it is a fundamental process because it accelerates the development and improvement of applied algorithms and software.

2.1.1 CPS Modelling Tools

In [16], the authors provide a broad survey about model-driven techniques and tools for generating CPSs. However, an important argument is not included, the compatibility with the Functional Mock-up Interface (FMI) standard [17] enabling generate Functional Mock-up Unit (FMU) models. The FMI interface is essential for development since it enables the usage of derived models in various environments and the connection of models generated with different tools. Almost all of the mentioned modeling tools also provide a simulation environment to perform the execution and tests. However, this comes with limitations when developing methods, algorithms, and tools requiring a different basic environment. In addition, standardized model formats such as FMI are a universal enabler for the execution of model simulation runs without the use of the initial modeling tools. Instead, standardized FMI libraries are utilized to simulate in co-simulation mode, which incorporates an integrated solver and step-by-step execution. In the following description, we set the focus on Python as the main programming environment since it provides an extensive database of libraries

[24] for data processing and ML or ANN algorithms. We make use of the FMU simulation framework [9] controlled via a Python interface based on the library called pyFMI [1]. The framework has been developed to execute simulations of CPS models in co-simulation mode. In addition, a user-friendly Python interface enables the loading, configuration, execution, and interaction with the CPS during simulation runtime.

2.1.2 CPS Model Validation

Validation of developed models is essential since this process provides information about the quality and usability of the generated data. We assume that at least measured data of normal system operation for different configurations is available. A prerequisite of the validation procedure is the execution of simulations for comparison. Therefore, we recommended copying the behavior and configuration of the measured data and executing simulation scenarios. This enables the assessment of the accuracy of representation in terms of behavior and dynamics, validated against real data. In essence, we show examples of potential processing methods for calculating similarity and distinguishability factors of different system configurations and conditions, also described as fault classes. It must be noted that our focus is on CPS with time series data output. Based on the output, we receive important information about potential model limitations that have a negative impact on the diagnosis algorithm. For the application of both methods, we suggest splitting time-series data into junks to avoid long-lasting shifts or drifts with the drawback of missing information about certain sections. However, the selected junks should display at least interesting behavior or shapes of the data, such as transition phases when new inputs are set.

Similarity Factor

A requirement for calculating a similarity factor is data preprocessing, including feature extraction, i.e., applying statistical methods and normalization. Normalization is important for directly comparing individual features and resulting similarity factors. A detailed description of the preprocessing is presented in Section 2.3 and in Algorithm 1. Figure 2 shows the application of several methods for a data snipped of time-series waveform type compared to modified versions. We focus on different methods like dynamic time warping (DTW) [26] [29], Euclidean distance, Manhattan distance, and Pearson correlation. The resulting matrix shows different sensitivities to the data and their modifications. Pearson correlation measures the linear relationship between two vectors of time series. Euclidean and Manhattan compute the straight-line distance error between two vectors with different approaches and are commonly used for comparing vectors or time series with regular intervals. DTW distance calculation is a valid option for performing validation tests to receive a factor about the similarity of data packages while accounting for tolerance of temporal distortions like shifts, stretches, and compressions between two sequences. DTW has main advantages when comparing time series with irregular intervals or different lengths. This aligns with most assumptions on datasets since measured and simulated data may not consistently synchronize with absolute accuracy. DTW distance in Figure 2 shows an overall good estimation of similarity, also for a slight time shift, where other measures like Euclidean distance apply a too-high weight on it. Pearson correlation could be used as an additional factor to estimate the linear relation between the signals.

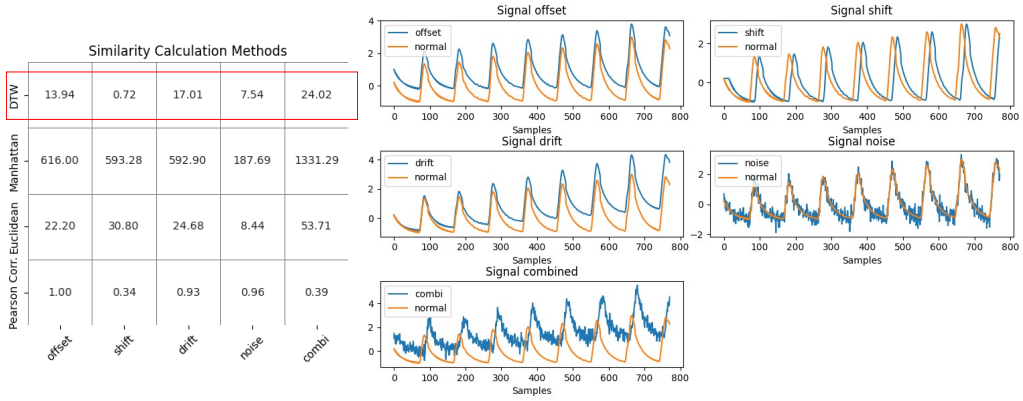


Figure 2 DTW distance computation applied on time-series data with waveform behavior. The figure shows an offset, phase-shift, and noise configuration and the corresponding DTW distance computation for each. offset: +0.8, shift: +2, drift rate: +0.02, Noise: std dev 0.4.

Distinguishablility Factor

Another important method regarding ML classifier models is the distinguishability (or isolation) factor of various fault classes. Thus, as mentioned earlier, the DTW method can be applied to measured or simulated data to analyze how fault classes can be isolated from each other. Figure 3 shows an example of multiple fault conditions in time-series data and a computed DTW distance matrix, which estimates the variation between the different fault conditions. Based on the matrix, we can argue that the classification of *Fault* 0 and *Fault* 1 can cause issues due to a relatively low factor compared to other data classes. The example in Figure 3 only compares one signal of different fault classes, but for CPS, we have to consider multiple signals of one class. Thus, we can also apply a dependent multi-dimensional DTW [29], capable of computing a distance between multivariate sequences. A concrete application is shown in the case study in Section 3.2.

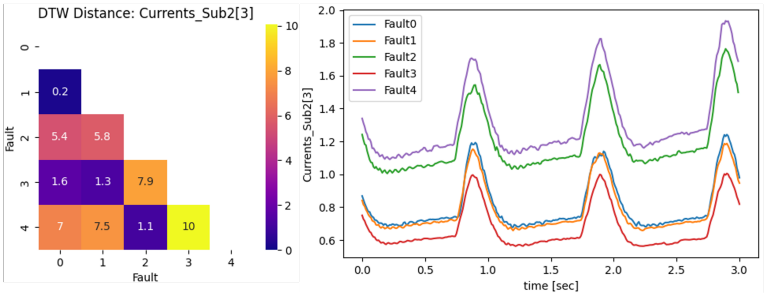


Figure 3 DTW matrix comparison between different fault configurations in the measured data for *Currents_Sub*2[3]. A value close to 0 means highly matching, and increasing values mean a deviation in the signals. The DTW matrix shows the signal for a load of 20 NM, a speed between 3000 and 3650 rpm, normal, and four fault configurations (F1 - F4). The data is standardized on a Z-score.

2.2 Simulation Data Generation

The data generation process based on the derived simulation models must be specified in detail before executing because the simulation runs may consume much time and performance. Thus, key questions are noted to support establishing relevant scenarios with high coverage

for the intended application of performing system diagnosis.

- How is the normal operational domain defined within the context of the system?
- What fault scenarios are known and documented?
- What fault scenarios within the system can be efficiently replicated using the CPS model?
- What fault scenarios are relevant to the CPS's fault detection process?

Based on these considerations, we define an appropriate fault case catalog tailored to the specified operational domain and the individual configuration setup. Simulation efforts may rise significantly with the increasing complexity of dynamic systems models. Therefore, countermeasures like discretization must be taken to reduce the number of simulation traces. For example, we assume that an e-motor model with a continuous controllable speed adjustment will result in an infinite number of traces to be recorded to capture the exact behavior in the data. Since this approach is not applicable for time and storage reasons, a discretization of speeds is required, which divides the speed profile into sections of interest, as shown in Figure 4. The selection of appropriate ML or ANN methods covers, at least to a certain extent, the missing areas between the selected discretization steps. The discretization level can be adapted based on the final fault detection accuracy evaluation to improve that. The next important step is elaborating scenarios and phases that represent a broad coverage of the model operation. In Figure 4 we illustrate different phases to be considered if qualified on the model output. We subdivide the parameter configuration into the following sequences:

- **Initial conditions:** To prevent overfitting in the data, the initial process should use different configurations for each simulation. If the model requires a specific ramp-up phase, e.g., an e-machine to establish a required speed, it is suggested that these sections be avoided by adapting the starting time of the observation recording accordingly.
- **Transition phases:** The transition phases are important since they could initiate critical scenarios or mask faulty behavior due to overproportional changes in the data compared to the failure's impact on the observation.
- **Temporal changes:** For the data acquisition, it is important to cover temporal changes and, depending on the required observation period, to adapt the sequence duration.
- **Equilibrium phases:** The identification of when a system reaches an equilibrium phase after a transition phase or a temporal change for early stopping or initiating new inputs is essential. This will prevent overfitting on these sections for later classifier model training.
- **Sequential occurrence:** The sequential injection of new inputs or faults is a good approach to save simulation time and storage, even so, it should be minded that a failure sequence should be injected in a normal system for correct representation, besides cases, where detection of consecutive failures is required to predict the propagation of such.
- **Fault injection timing and duration:** The timing and duration of fault injection are important to maximize the scenario coverage. Our experience showed that a fault should be injected into a normal system when it reaches the equilibrium phase of the actual configuration (if applicable to the model). With this, scenarios of different faults can be compared based on their impact on signal behavior. The scenarios can further be improved in terms of coverage by setting different system or component states during the presence of the fault.

It is clear that not all scenarios, for example, initialization of faults during each timestep of a long-lasting transition phase, are achievable. However, a structured selection of scenarios may overcome such limitations in combination with an appropriate classifier model. After specifying the relevant phases, a parameter configuration is defined for an efficient training data generation process. Annotated contextual information is required to achieve a labeled training dataset. Therefore, configuration settings, system states, and conditions are logged

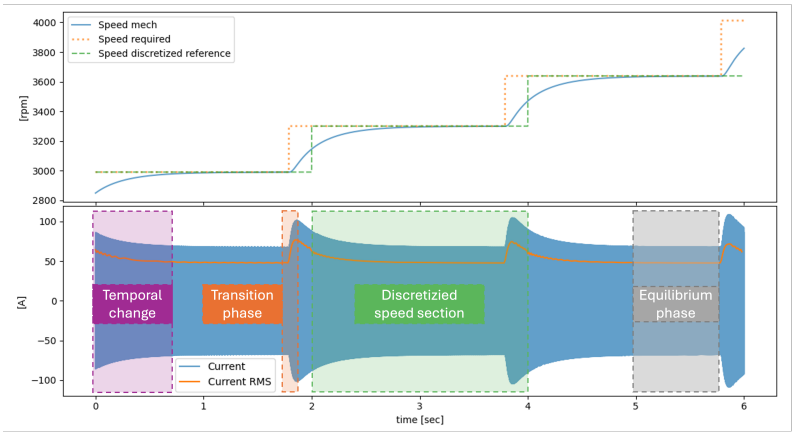


Figure 4 Identification of phases in signals and observations of a CPS.

together with the observed system inputs and outputs when executing simulations. For execution of simulations, we utilize the tool [9]. It provides an interface to extract a data package with the collected input and output data and the corresponding label information in a time-series format.

2.3 Data Preprocessing Methods

Before applying a classifier algorithm, the generated simulation training data requires preprocessing to extract features to determine different configurations and failure modes in the data. Based on the received data, corresponding methods are applied as discussed in [4], where the authors describe a comprehensive review of the overall procedure starting from data cleaning, reduction, scaling, transformation, and partitioning. When following this structure, the initial task is the cleaning procedure, which is mainly separated into two topics, value imputation and outlier elimination. Since we use generated simulated data, this step can be neglected. The data reduction is applied to the rows by resampling at equal sampling rates or removing duplicate sections within the same configurations. Alternatively, it can be applied to columns, which the authors in [4] divide into three domains, the expert knowledge feature selection, the statistical feature selection, and the feature extraction methods for computing enhanced data properties. The feature selection based on domain expert knowledge shall be preferred if applicable. However, with an increasing number of measured signals, the computed statistical method supports identifying correlated signals and provides information about the variance in the dataset. Another important task is the application of feature extraction techniques to generate new features through linear or nonlinear methods. The analysis of linear data patterns can be achieved by using statistical techniques and Principal Component Analysis (PCA) [5]. Both approaches simplify complex information by combining or transforming data in linear ways. Another approach is nonlinear feature extraction, which has the advantage of minimizing the potential threat of information loss when new features are extracted based on identified patterns in the dataset. Such a method are, e.g., autoencoders (AE) [28]. Regardless of whether PCA or AE is used for complexity reduction, the statistical method remains an appropriate and simple preprocessing step for feature extraction and data size reduction. Since we focus on the time-series domain, implementing a moving time window method is reasonable for computing statistical values like the mean, Root-Mean-Square (RMS), variance, and others. Since RMS is relevant for the case study

presented later, its definition is provided as RMS $= \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$, where x_i represents the individual values, and n denotes the total number of values. Data normalization is essential to generalize each feature equally to receive a homogeneous dataset. In [22], the authors discussed several normalization methods that imply scaling or transforming data to highlight the impact on simple ML classifier performances. The results showed that Z-score and Pareto Scaling performed well for most tested datasets based on the full and selected feature set. The authors also stated that choosing the proper normalization method depends on the quality of the dataset. When using distance-based classifier algorithms, noise or outliers can lead to stronger responses with a negative impact on the performance.

The last step references the labeling of preprocessed data for the later supervised learning algorithms. Therefore, it is important to define clear labels as integer values representing different contextual information. This approach should be considered for classifiers with single- and multi-output predictions.

2.4 Classifier Models

Shakiba et al. [21] provide comprehensive research and evaluation on ML algorithms focusing on solving fault diagnosis problems in the domain of electrical power systems. The survey supported our selection process for the most appropriate models regarding applicability and straightforwardness in implementation, which are the Decision Tree (DT), Random Forest (RF), k-Nearest Neighbor (k-NN), and Multi-Layer-Perceptron (MLP). In addition, we will discuss a simple design of an unsupervised Autoencoder (AE) [27] in the later introduced case study. In general, AEs support dimensionality reduction, feature learning, time series forecasting, or anomaly detection based on their model design. For anomaly detection, AEs are trained on normal system data to predict the reconstruction of a given observation. An anomaly is detected when the reconstruction error between the learned behavior and observation exceeds a defined threshold. Although this approach does not provide information about the root cause of a fault, it improves a fault detection method by identifying unknown and unexpected system behavior. This led to the idea of utilizing an AE for anomaly detection in combination with an ML classifier to overcome the limitations of both options. A concrete implementation of the idea is presented in Section 3.5 and Algorithm 2. In this paper, we focus on 1D vector data of m feature values as input for the models. We do not consider a time sequence of n samples and m features as input for the ML classifiers or the AE.

3 Case Study

In this section, we present an application of the general approach based on a case study comprising a dual three-phase machine. First, we provide a brief introduction to the model and specifications. Second, we follow the processing pipeline, starting with the CPS model validation by calculating the similarity between the simulated and measured data and the distinguishability of several investigated system conditions. Next, we discuss the simulated data generation and the preprocessing implementation to prepare the data for the diagnosis part, followed by the concrete application of an AE for anomaly detection and an ML classifier for fault identification in real measured data. Finally, we discuss the results and show potential applications, drawbacks, and limitations.

3.1 CPS Model Background and Specifications

According to [18], common faults in electric drives are related to stator winding faults. These faults account for 36% for low-voltage and up to 66% for high-voltage machines. Stator winding issues arise due to insulation stress, material degradation, vibrations, temperature, and high voltage stress. The most typical fault is an inter-turn short-circuit (ITSC), which usually initiates as a single-turn short-circuit and spreads rapidly to the entire winding, resulting in a ground fault. This causes a severe emergency and consequently forces an operational stop. The study focuses on dual three-phase electric motors operating under different ITSC faults. The primary goal is to evaluate a diagnosis algorithm to detect these faults in the time-series domain during runtime using synthetic-generated data within a simulated environment. Specifically, we want to validate if and to what extent simulated data can be used to detect faults with root cause analysis on real measured data. Thus, we utilize the dual three-phase e-motor model as introduced by [13]. In addition to the proposed simulation model, real hardware measurements, including ITSC fault-injected data, are accessible. Details of the simulation model and the experimental hardware setup are presented in [15], [13], and [14]. Table 1, and 2 provide an overview of the specifications, available signals, and selected operational domain for the diagnosis procedure evaluation. The system's controller is designed to align the actual rotational speed, ω_{mech}, with the reference rotational speed, ω_{ref}, by adjusting the control signals, which are the voltages in DQ coordinates. These adjustments directly impact the measured currents in the phases. Further, the model can simulate various faults focusing on ITSCs with different configurations, as shown in Table 3. We divide the addressed ITSC faults into two schemas: Fault 1-6, which provides information on the affected phase and the number of shorted turns in the winding, and Phase Fault A and B, which categorizes faults based solely on the affected winding phase. The authors of the model used Matlab Simulink [6] as a modeling tool and provided an exported FMU model in co-simulation format to fit into our developed FMU simulation environment [9].

Table 1 Dual three-phase e-motor parameters and configuration.

Configuration Parameter	**Range**	**Unit**
Nominal configuration		
DC voltage	200	V
Max. continues motor current	107	A
Winding resistance	6.5	$m\Omega$
Winding inductance	180	μH
Nominal speed	8000	rpm
Maximum speed	10500	rpm
Nominal power	30160	W
Number of pole pairs	10	
Test setup		
Test speed range	0 to 5000	rpm
Test load torque	0 - 35	Nm

Table 2 Dual three-phase e-motor measurement output signals.

CPS Signal Name	**Reference Name**	**Unit**	**Symbol**
Speed Mechanical	Speed_mech	rpm	ω_{mech}
Speed Reference	Speed_Reference	rpm	ω_{ref}
Current Subsystem 1 (A, B, C)	Current_Sub1 ([1], [2], [3])	A	I_{1a}, I_{1b}, I_{1c}
Current Subsystem 2 (A, B, C)	Current_Sub2 ([1], [2], [3])	A	I_{2a}, I_{2b}, I_{2c}

Real Measured Data

As already mentioned, we have two models, the real hardware and the simulation model. Both provide currents and voltages of each subsystem and phase, resulting in six different currents. Also, currents and voltages in DQ coordinates are accessible. We decided to focus on phase currents, I_{1a}, I_{1b}, I_{1c}, I_{2a}, I_{2b}, and I_{2c}, which can be easily measured even in already existing systems. Another signal considered is the system's measured rotational mechanical speed ω_{mech}. The torque load τ is not measured with the hardware setup. Instead, an average target torque is provided as additional information, which is later used to evaluate the applied diagnosis methods. All signals are recorded at a sampling rate of 10 kHz. The setup provides a fault injection interface capable of simulating ITSC faults, as shown in Table 3. The available measured data has the following configuration:

- stepwise increasing ω_{ref} changes from 10 rpm up to 4700 rpm,
- the stagnant duration for each ω_{ref} is 1.0 seconds,
- torque load τ from 0 Nm to 35 Nm with a stepsize of 5 Nm,
- normal and fault system conditions (see Table 3),
- total execution time is 54.0 seconds for each configuration.

We received a total number of 56 measured cases, each lasting 54.0 seconds and including information about I_{1a}, I_{1b}, I_{1c}, I_{2a}, I_{2b}, I_{2c}, and ω_{mech}. In addition, configuration settings, like the reference speed ω_{ref}, injected fault condition, and actual load factor τ are stored. The data packages are stored as Matlab .mat format files, each with a size of 25 MB. The measured data packages are publicly accessible on Zenodo [11].

3.2 CPS model validation

For the presented case study, we have access to an extended set of real measurement data for the normal and different ITSC fault injection scenarios. To achieve a precise comparison, we extract the different applied configurations from the measured dataset to reproduce a similar dataset within the simulated environment. With the presence of the two datasets, measured and simulated, we initiate a procedure to validate similarity and distinguishability. Both metrics are separated into a visual and calculated part. For the visual part, a graphical representation of signals is plotted to display obvious discrepancies like offsets, shifts, delays, or inaccurate behavior. We use the DTW distance method based on the Python library [29] for the calculated part. The configuration of the datasets shows a stepwise increasing ω_{ref} (from 100 to 4400 rpm) where each step is held for approximately 1.0 seconds until the next reference speed ω_{ref} is set. For the CPS validation, we use these speed steps as discretization levels for the different stages of the data to split into subsamples, for example, see Figure 4. With this, a meaningful DTW can be calculated, and an implication on speed effects is achieved. An essential part is data preprocessing, which extracts features that show relevant

Table 3 Dual three-phase e-motor inter-turn short-circuit fault configurations.

Normal	**Description**	**Shorted turns**	**-**
Fault 0	Normal operation of a system	0/7	Fault 0
Turn Fault	**Description**	**Shorted turns**	**Phase Fault**
Fault 1	ITSC in phase B in sub-system 2	1/7	Fault B
Fault 2	ITSC in phase A in sub-system 2	2/7	Fault A
Fault 3	ITSC in phase B in sub-system 2	3/7	Fault B
Fault 4	ITSC in phase A in sub-system 2	4/7	Fault A
Fault 5	ITSC in phase B in sub-system 2	5/7	Fault B
Fault 6	ITSC in phase B in sub-system 2	6/7	Fault B

information about the different system conditions (faults). Because the different ITSC faults cause an amplitude change in the corresponding currents, a moving time window with the statistical RMS computation is applied to reveal the deviations. Since the measured currents' waveform frequency is linear increasing with ω_{mech}, we might not be able to extract the relevant information (RMS amplitude) for low ω_{mech} if the time window is selected too small and does not cover at least one full period. On the other hand, selecting a too-large time window is problematic for high ω_{mech}, where we might miss important information. Thus, we introduce the adaptive time window based on the actual ω_{ref} to cover a defined number of periods in the data (see Algorithm 1, with $n_{poles} = 10$ for the motor specific pole pairs, $n_{periods} = 3$ for three periods and $n_{sampling} = 10000$ for 10 kHz frequency data recording). In the upper right corner of Figure 5, the different windows for the selected discretized ω_{ref} values are shown, and Algorithm 1 provides details about the concrete implementation.

Similarity Visualization & Measurement

Figure 5 illustrates a similarity computation between the measured and simulated data. The first two plots show the signal data for I_{2a}, I_{2b}, and I_{2c}. The left plot shows the original data and the right plot shows the preprocessed data with the adaptive moving time window and RMS and Z-score normalization. The adaptive window mechanism is based on the discretized speed value ω_{ref} and the stepsize Δs is 100 (see Algorithm 1). The plots reveal obvious correlation issues regarding amplitude peaks during speed changes and the response behavior. The data shows an acceptable convergence when a static speed phase is reached. The seven plots below illustrate the DTW distance matrix of each signal based on τ and ω_{mech} factors. The white boxes indicate the data represented in the corresponding discussed signal graphs. From the plots, we measure an increasing offset drift together with increasing ω_{mech} between the measured and simulated data signals of I_{2b} and I_{2c}. But a more stable and accurate representation is measured for I_{2a}. The overall analysis of the current signals for a normal operation shows an acceptable representation with a low DTW distance between 0.6 and 3.6 with a trend towards higher ω_{mech}. For the individual fault data comparison, we see an increasing DTW distance at areas with high τ (30 - 35 Nm) and ω_{mech} (3000 to 44000 rpm). Also, areas with a τ between 0 and 5 Nm and ω_{mech} above 1500 rpm indicate a lower similarity for some signals. The overall DTW distance for fault conditions is between 0.08 and 6.12. The DTW distance values as numbers do not provide direct information but act as indicators of lower similarity in areas or segments within overall higher averaged values.

Algorithm 1 Preprocessing algorithm.

Input: Raw data $\mathbf{D}_{raw} = \{(X, y)_{\tau_0, f_0} \dots (X, y)_{\tau_k, f_l}\}$ with n samples, k different τ and l different classes ($0 \equiv$ normal system, 1-$l \equiv$ faulty system). Δs as step size for moving time window method (**MTW**), n_{poles} is the pole pairs, $n_{periods}$ is the captured periods, $n_{sampling}$ is the raw data sampling frequency , F list of features, L list of labels, *metric* a list of statistical methods.
Output: Datapackage D with labels and RMS normalized features and normalization parameters N

function PREPROCESSING(D_{raw})
  $X, y \leftarrow D_{raw_{(\tau_{0\to k}, f_{0\to l})}}$ ▷ extract F and L data from raw data for all k τ and all l system classes f
  $\Delta w(\omega_{ref}) = \lfloor \frac{120}{\omega_{ref} * n_{poles}} * n_{periods} * n_{sampling} \rceil$ ▷ time window related to ω_{ref} to capture $n_{periods}$ periods in waveform data (currents)
  $X_{stats} \leftarrow \mathbf{MTW}_{0\to n}(X_\omega, \Delta w(\omega_{ref}), \Delta s, F, metric) \quad \forall X_\omega \in X(\omega_{ref})$ ▷ ω_{ref} extracted from data to compute corresponding Δw and X_{stats} with $n_{stats} = \frac{n}{\Delta s}$ samples and $n_F = len(F) * len(metric)$ features
  $y_{stats} \leftarrow \mathbf{MTW}_{0\to n}(y_S, \Delta s, L)$ ▷ L labels with Δs with $n_{stats} = \frac{n}{\Delta s}$ samples
  $X_{stats,norm}, N \leftarrow$ normalization of X_{stats} ▷ N normalization parameters
  return $D \leftarrow X_{stats,norm}, y_{stats}, N$
end function

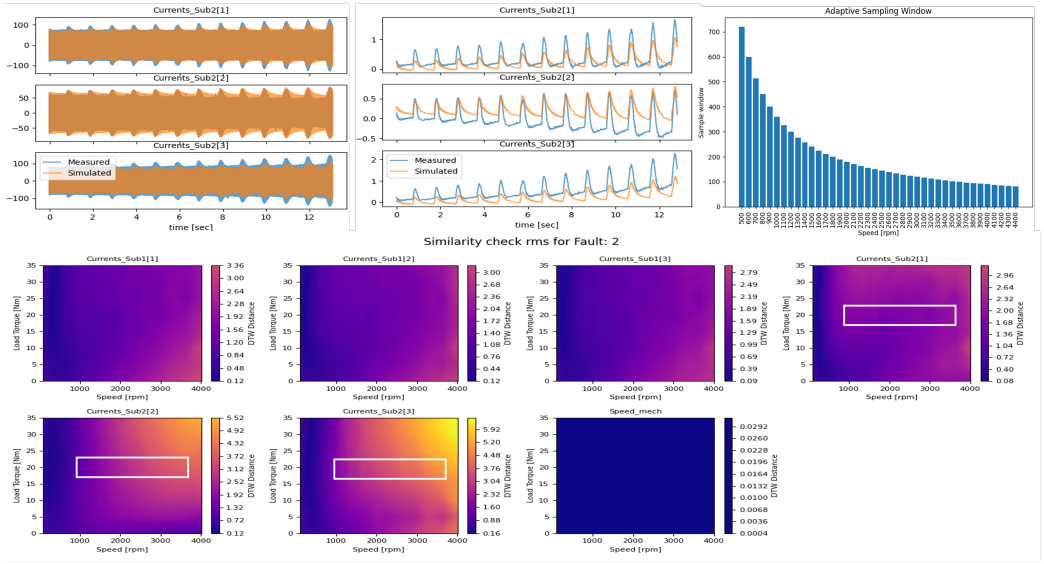


Figure 5 DTW comparison between simulated and measured data for each feature of Fault 2. Low values indicate similarity and high values indicate deviation. The DTW matrix shows three signals, each for τ from 0 to 35 Nm and ω_{mech} from 1000 to 3650 rpm. The signals in the upper graphs represent τ 20 Nm and ω_{mech} from 1000 to 3650 rpm. The data is normalized with a Z-score.

Distinguishability Visualization & Measurement

The distinguishability factor verifies whether and how strong the system conditions (normal or faulty) differ in shape and offset within the measured, simulated, or between the data types. In Figure 6, we present the distinguishability results computed on the reference of measured data compared with the equivalent conditions in the simulated data. The system conditions start from normal (Fault 0) to Fault 1-6. We utilize the multi-dimensional DTW algorithm presented in [29] for the calculation. The results show the computed DTW distance in correlation with ω_{mech}. Each line represents the DTW distance between the reference measured fault state and all other simulated fault states. The results reveal a very similar behavior of features for low ω_{mech} (< 500 rpm). Hence, fault detection might not be possible for scenarios where ω_{mech} is below 500 rpm. The first plot in Figure 6 compares the measured normal operating system with all other simulated fault states. The shadows represent the minimum and maximum values for the different applied loads τ. The results show that normal measured system behavior can be distinguished from simulated faulty scenarios with a relatively good estimation, except for Fault 1, which overlaps with high similarity in the behavior. Fault 1 is the weakest ITSC fault because only one shorted turn in the winding is simulated. Thus, we can conclude that Fault 1 will lead to classification issues in the later applied ML models. Further, Fault 2 shows a weak distinguishability from Fault 4. The same applies to Fault 3, Fault 5, and Fault 6. Important to mention is that the plots reveal a relatively high distinguishability between faults located at winding phases A and B.

3.3 Simulation Data Generation

This section explains the simulation data generation process based on the CPS model. The target is to receive labeled datasets with broad scenario coverage, including normal and faulty system behavior representations. To achieve that, we adhere to the following positions:

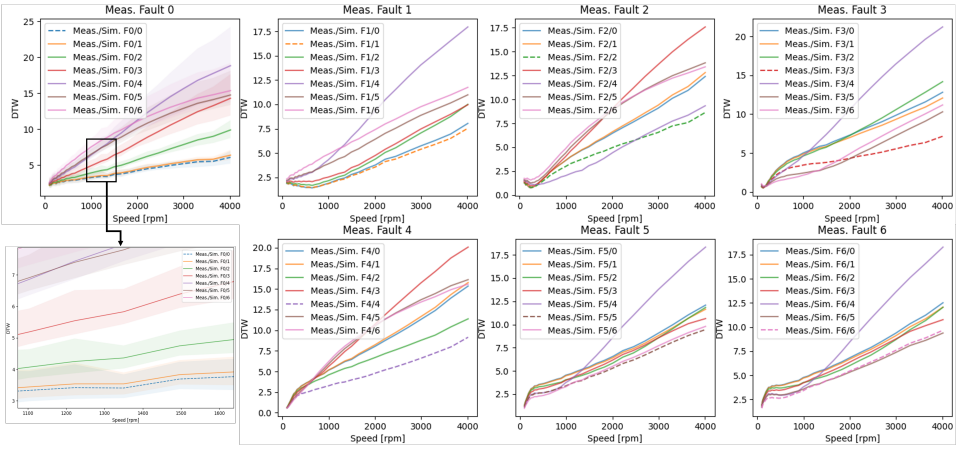


Figure 6 DTW distinguishability between measured (reference) and simulated normal and Fault 1-6 data for all features combined. The features are used on τ 0 - 35 Nm (shadowed area indicates min and max values) and ω_{mech} from 100 up to 4400 rpm. The data is normalized with Z-score.

- **Definition of operational domain:** We focus on the test setup as defined in Table 1 for a speed range from 0 to 5000 rpm and a load from 0 to 35 Nm.
- **Discretiation of parameters:** Since the model offers two main continuous parameters, the ω_{ref}, and applied τ, we discretize them to reduce the number of required simulation runs, aiming to maintain high scenario coverage despite the reduction in simulation runs. For ω_{ref}, we apply a simple step function to receive a value range between 100 and 5000 rpm with a stepsize of 100. For τ, we apply a step of 5 Nm for a value range of 0 to 35 Nm.
- **Specification of important phases and scenarios:** This process involves a clear definition and outline of essential phases and scenarios crucial for understanding and testing a system. During the model validation, we already noticed that the simulation model contains a delayed response when representing the dynamics of the currents after a ω_{ref} change. Selecting a suitable period for simulating each ω_{ref} step is necessary to account for this effect. This includes the transition phase (ω_{ref} is changed), the temporal change phase, and the equilibrium phase. An illustration of these sections was previously presented in Figure 4. We simulated 2-second scenarios for each ω_{ref}. With the selected period, we cover the whole transition and temporal change phase and a part of the equilibrium phase, but to a moderate extent to avoid overfitting.
- **Simulation configuration and execution:** The simulation framework and model specification setup comprises the specific conditions, initial states, input states, and simulation parameters definition to represent real-world scenarios and generate meaningful, accurate outputs. The defined operational domain and the consideration of explored system phases enable extracting information about the affected parameters and their value range to cover a large set of scenarios for the data generation process. The collected outputs are the currents I and ω_{mech}. The labels (classes) are integer values related to normal operation (0) and the ITSC faults (1-6). Multiple simulations are executed based on all relevant combinations of the parameter values as initial states, so we do not consider a fault injecting during runtime. Only the reference ω_{ref} is adjusted to iterate through the discretized range at intervals of 2.0 seconds. Each simulation configuration generates outputs with a sampling rate of 10 kHz and a total time of 100.0 seconds. The simulation framework, as presented in [9], is utilized, serving as a robust platform to

execute simulations and generate the labeled output data. We generated 56 scenarios (τ 0 - 35 Nm and Fault 0 - 6), each with a simulation time of 100 seconds and a file size of 242 MB. Approximately 280 hours were required to simulate the full data package by utilizing the computer setup[1].

3.4 Data Preprocessing Method

The preprocessing method is important since it significantly impacts the performance of the applied diagnosis methods. As discussed in Section 2.3, we follow the guided structure, beginning with feature selection, feature extraction, data normalization, labeling, and partitioning. For the dual three-phase model, we select the currents (I) and rotational speed (ω_{mech}). We apply the statistical method RMS to extract features from the given data packages, which already showed reasonable results during the distinguishability validation. Other methods like mean, variance, or slope do not produce valuable features because the system behavior for the fault conditions is mainly represented as amplitude changes in the currents. A specific size of a data window is required to calculate the RMS. Thus, the discretized speed sections represented as ω_{ref} are utilized to calculate the time window to keep it constant for specific sections (see Algorithm 1). A stepsize Δs of 100 samples is used to move the window through the time-series data to minimize duplicates and reduce the sample size. Because of a manageable number of available features, PCA and AE do not provide benefits in terms of dimensionality reduction, in fact dimensionally reduction even results in lower accuracy for the fault classification. For the normalization, the Z-score method is applied. All preprocessing steps performed on the simulated data are also applied to the measured data, but with the difference that according to a real system structure, each timestep update is separately received. To apply the moving time window approach, the data is collected until the required window size for the given ω_{mech} is acquired. The time window procedure follows the first in, first out principle. The stepsize Δs is used as a trigger to initiate a diagnosis on equally distributed intervals. When a diagnosis is triggered, the statistical method RMS is computed, and the normalization of the results is performed on scaler parameters as utilized for the simulation data. This enables a general valid scaling with equalized weighted features. Details are shown in Algorithm 3 from lines 3-9.

3.5 Classifier Models

In total, three datasets are available for evaluating the ML classifiers. The datasets include the simulated training data, the measured test data, and the simulated test data, which is a simulated copy of the measured test data. Table 4 shows different classifier models utilized on three options, either solely simulated or measured data or the interesting part simulated training data and measured test data. The ML model accuracy is evaluated on cross-validation with a test data split of 30 %. The results show that models trained on measured data are applicable to classify the normal and every single fault with high accuracy. This highlights the potential of measured data for fault identification and clearly demonstrates the effectiveness of the preprocessing method, as it enables accurate fault detection. We see a low prediction accuracy for the identification of the exact shorted turn fault when using simulation data for training and measured test data. However, detecting a fault-affected phase (Fault A or B) can be done with a high accuracy of 98% for RF models

[1] We use a MacBook Pro (2021), Apple M1 Pro, 16 GB memory, macOS Ventura 13.2.1, and Python 3.9.

Table 4 ML classifiers tested on simulated, measured, and both options. ω_{mech} ranges from 500 to 4400 rpm and τ from 0 to 35 Nm. Faults 0-6 are used. Grid Search (GS) is used with cv=5 and a test data split of 30 %. ML model default parameters [19] with adaptions for GS: DT: max depth=[50, 100, 200]; RF: n estimators=[100, 150, 200]; k-NN: n neighbors=[1,3,10,50,100]; MLP: hidden layers= [(30,), (30,10,30), ...], alpha=[0.0001 - 0.01], solver=[adam], activation=[relu, tanh].

Train	Test	Model	F 0,1-6		F 1-6		F 0,A,B		F A,B	
			Z-score	min-max	Z-score	min-max	Z-score	min-max	Z-score	min-max
Sim.	Sim.	RF	0.874	0.876	0.905	0.905	0.949	0.948	0.985	0.984
		k-NN	0.872	0.854	0.936	0.924	0.908	0.903	0.979	0.976
		DT	0.834	0.839	0.859	0.860	0.923	0.924	0.968	0.967
		MLP	0.975	0.944	0.995	0.999	0.983	0.970	0.999	0.998
Meas.	Meas.	RF	0.919	0.919	0.910	0.910	0.998	0.998	0.999	0.999
		k-NN	0.920	0.910	0.912	0.900	0.998	0.996	0.999	0.999
		DT	0.881	0.881	0.871	0.872	0.991	0.991	0.996	0.996
		MLP	0.984	0.973	0.973	0.969	0.996	0.998	0.998	0.998
Sim.	Meas.	RF	0.429	0.427	0.478	0.488	0.834	0.832	**0.982**	**0.982**
		k-NN	0.451	0.446	0.505	0.502	0.809	0.800	0.952	0.945
		DT	0.365	0.367	0.422	0.416	0.823	0.826	0.932	0.933
		MLP	0.435	0.333	0.455	0.448	0.850	0.854	0.963	0.976

and both normalization methods. Having this in mind, and the fact that the normal data generated by the CPS model is represented with an acceptable similarity to the real system, a tradeoff to the initial planned diagnosis approach is made by leveraging an unsupervised Autoencoder (AE) model for anomaly detection [27] in combination with a Random Forest (RF) classifier. Algorithm 2 provides details about the implementation of the AE and RF methods, and Algorithm 3 shows the application on measured data. The AE configuration consists of an encoder and a decoder. The encoder compresses input data using a single dense layer with linear activation. The decoder reconstructs the original input from the encoded representation using another dense layer with linear activation. During training, simulated data is used as the training dataset, while measured data is used for validation. The model minimizes the mean squared error between the input and its reconstruction. An anomaly is detected if the reconstruction error (RE) between the input and the predicted reconstruction exceeds a threshold. The threshold is derived by utilizing the RE of normal measured data (lines 7-11). Lines 7-11 show the ω_{mech} dependent threshold function calculation. The RF model is trained on simulated fault data comprising labels A and B.

3.6 Results

In this section, we show details about the results of the AE implementation for anomaly detection in combination with an RF classifier (default configuration with estimator n = 200) to detect the faulty phase in the dual three-phase e-machine. The data preprocessing uses RMS as a feature and Z-score for normalization. Figure 7 shows the prediction accuracy of the complete application tested on measured data with normal and Fault 1-6 behavior for τ from 0 to 35 Nm and ω_{mech} from 500 to 4400 rpm. We achieve a 100 % accuracy rate in identifying normal system behavior. Only minor undetected or misclassified anomalies exist for Faults 2-6 and τ below 5 Nm. Detecting anomalies for Fault 1 proves challenging, as noted with the distinguishability factor. However, at speeds ω_{mech} higher than 1200 rpm and loads τ greater than 15 Nm, high prediction accuracy is also achieved for Fault

Algorithm 2 Diagnosis Methods.

Input: Simulated preprocessed annotated data D_S and measured preprocessed annotated data D_M. Threshold factor r to tune the sensitivity of anomaly detection.
Output: Trained AE model $\mathbf{A}$, threshold $\Theta(\omega_{ref})$, trained ML model $\mathbf{M}$

function TRAIN DIAGNOSIS METHODS(D_S, D_M)
  $X_{S_0} \leftarrow D_{S_{(\tau_{0\to k}, f_0)}}$ ▷ extract normal simulated data for all τ
  $X_{S_f}, y_{S_f} \leftarrow D_{S_{(\tau_{0\to k}, f_{1\to l})}}$ ▷ extract fault injected simulated data for all τ
  $X_{M_0} \leftarrow D_{M_{(\tau_{0\to k}, f_0)}}$ ▷ extract normal measured data for all τ
  $\mathbf{A} \leftarrow$ trained AE anomaly detection model on X_{S_0} and validated on X_{M_0}
  Compute threshold on measured data: $\hat{X}_\omega = \mathbf{A}(X_\omega) \quad \forall X_\omega \in X_{M_0}(\omega_{ref})$ ▷ where X_ω is a matrix with n samples and m features, ω represents a feature in X_{M_0}
  Compute **RE**: $\varepsilon_\omega = RMSE(\hat{X}_\omega, X_\omega)$ ▷ **RE** is the reconstruction error
  Compute threshold: $\Theta_\omega = \mu_{\varepsilon_\omega} + r \cdot \sigma_{\varepsilon_\omega}$ ▷ threshold dependent on ω and factor r
  with: $\mu_{\varepsilon_\omega} = MEAN(\varepsilon_\omega) \quad , \quad \sigma_{\varepsilon_\omega} = STD(\varepsilon_\omega)$
  Compute cubic polynomial coefficients using $c = f(\Theta_\omega, \omega, 3)$
  Create polynomial function $\Theta(\omega) = f_{poly1D}(c)$ ▷ the function $\Theta(\omega)$ returns the Θ_ω based on ω
  $\mathbf{M} \leftarrow$ trained ML classifier model on X_{S_f} , y_{S_f}
  return $\mathbf{A}, \Theta(\omega), \mathbf{M}$
end function

Algorithm 3 Diagnosis Application.

Input: Measured test data $\mathbf{D}_M = \{X_{M_{\tau_0, f_0}} \dots X_{M_{\tau_k, f_l}}\}$, Trained AE model $\mathbf{A}$, threshold $\Theta(\omega)$, trained MLP model $\mathbf{M}$, normalization model N_S fitted on simulated data
Output: Diagnosis $\hat{y}_i$

procedure DIAGNOSE APPLICATION($D_{M_{raw}}$, $metric$, Δw, Δs, N_S)
  Load $\mathbf{A}$, $\Theta(\omega)$ and $\mathbf{M}$
  $X_M \leftarrow D_{M_{raw}}$ ▷ X_M holds all raw feature observations
  for $i \leftarrow \Delta w$ to length of $X_M - 1$ by 1 **do**
    $X_{\Delta_i} \leftarrow X_{M_{i-\Delta w \to i}}$
    if $i \mod \Delta s = 0$ **then**
      $\vec{x}_{stats} \leftarrow metric(X_{\Delta_i})$ ▷ apply feature extraction by computing metric for each feature in X_{Δ_i}
      $\overline{\omega} \leftarrow mean(X_{\Delta_i, \omega_{mech}})$ ▷ extract the window data of feature ω_{mech} and compute the $mean$
      $\vec{x}_{stats,norm} \leftarrow N(X_{stats})$ ▷ compute normalization with N_S parameters
      $\mathbf{D}$ = DIAG($\vec{x}_{stats,norm}$,$\mathbf{A}$, $\Theta(\omega)$, $\mathbf{M}$) ▷ execute diagnosis approach
    end if
  end for
end procedure
function DIAG($X_{stats,norm}$, $\overline{\omega}$, $\mathbf{A}, \Theta(\omega), \mathbf{M}$)
  $\vec{x}_{stats,norm}$ with shape $(1, n_F)$ ▷ $n_F = length(features) * length(metric)$ and 1 sample
  $\overline{\omega}$ is the actual measured rotational speed
  $\hat{x} = \mathbf{A}(\vec{x}_{stats,norm})$ ▷ predict the feature values for normal data
  if RE$(\hat{x}, \vec{x}_{stats,norm}) > \Theta(\overline{\omega})$ **then** ▷ anomaly detected, fault identification
    $\hat{y} = \mathbf{M}(\vec{x}_{stats,norm})$ ▷ apply RF to predict fault class
    if $\hat{y}$ probability $> 60\%$ **then** $DIAG = \hat{y}$ ▷ $\hat{y}$ holds the phase fault code $\hat{y} \in \{A, B\}$
    else $DIAG = unknown\ fault$
    end if
  else $DIAG = normal$ ▷ no anomaly detected
  end if
  return $DIAG$
end function

1. The highlighted section in Phase Fault A and B provides a detailed view of the fault state, ω_{mech} (1500 - 1700 rpm), τ (0 - 15 Nm), and the reconstruction error observed for a normal system, Fault 1, and Fault 2 injected systems. The red tags indicate where the AE missed detecting an anomaly (false negatives), while the yellow tags denote cases of false classification by the RF model. The confusion matrix provides a summary of the overall classification outcomes. Based on this, we conclude that employing this approach is suitable for ω_{mech} from 500 to 4500 rpm. While the tests are conducted solely up to rotational speeds of 4500 rpm, the data indicates that the algorithm can be effectively applied at even higher speeds when utilizing the actual simulation model. Regarding time performance, we see disadvantages compared to the system's relatively high sampling frequency in the diagnosis process. Due to the adaptive moving time window method, a specific sampling window is required for different ω_{mech}. Windows for low speeds of around 500 samples represent 0.05 seconds for a sampling frequency of 10 kHz. For higher speeds, the value decreases below 0.01 seconds. This timeframe is only relevant to ensure the window fully covers a fault. The implementation of the AE is not optimized in terms of speed performance, thus a prediction takes 0.018 seconds. The RF model is relatively fast, with a prediction time of 0.004 seconds. Consequently, a diagnosis takes 0.022 seconds plus the active window when executed[1].

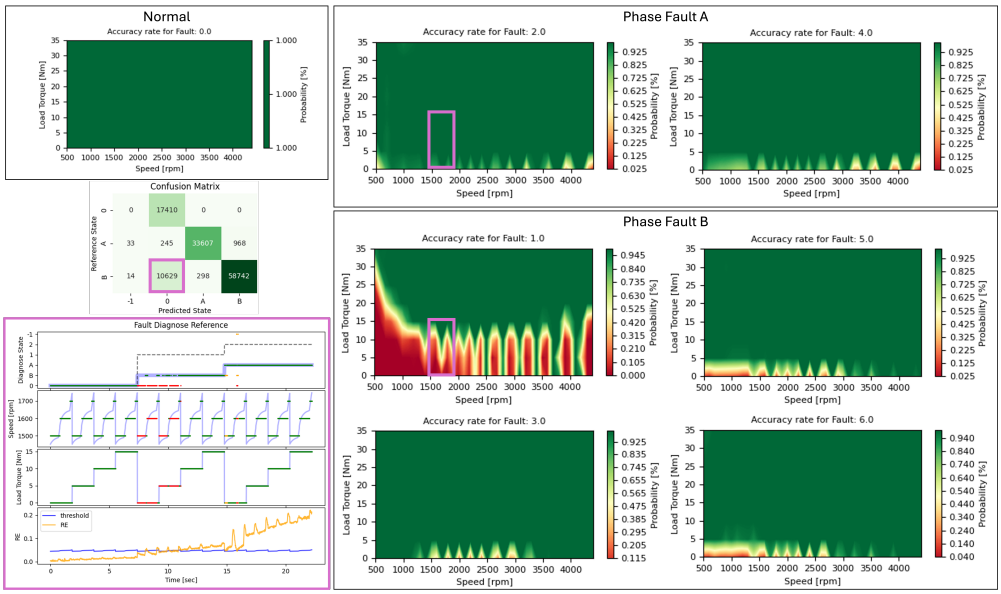


Figure 7 Diagnosis results with an anomaly detection (AE) and sequentially applied RF classifier model trained on Fault Phases A and B. Measured data with normal and Fault 1-6 with τ 0 to 35 Nm and ω_{mech} 500 to 4400 rpm is used as test data.

4 Conclusion

In conclusion, our paper presents a comprehensive simulation-based approach to diagnose a CPS. We utilize simulation models of the CPS to generate training data for ML classifiers applied for fault detection with root cause analysis. Our general approach comprises various stages, beginning with model validation, simulation data generation, data preprocessing, and finally, the implementation of fault detection models. In our case study, which focuses on a dual three-phase e-machine, we demonstrated the challenges of this approach. Measurements

of a real system under various conditions, including normal operation and operation with an active inter-turn short-circuit, are used to evaluate the implemented methods. A detailed simulation model offers a powerful and complex system representation including the capability to simulate inter-turn short-circuit faults. The presented preprocessing method enables the extraction of details from the observed signals by applying an adaptive moving time window to calculate the root-mean-square. In the CPS model validation procedure, limitations in terms of signal similarity and distinguishability between the measured and simulated data are revealed. Thus, the focus is on detecting the affected phase rather than the exact number of shorted turns in each phase as initially planned. It is worth mentioning that the detection of the affected phase is also important because selecting a valid mitigation strategy initially depends on identifying the specific phase. However, the diagnosis information about fault severity (shorted winding turns) is beneficial for adapting the chosen mitigation parameters to achieve a more fine-tuned risk reduction. As a result, an AE for anomaly detection is implemented to ensure the identification of unexpected behavior. A positive detection initiates a second layer, an ML classifier (RF), to identify the faulty phase. The approach shows high accuracy in anomaly detection of the phase with the shorted turns, except for single shorted-turn faults, which indicate a weak distinguishability with normal operation. Thus, no anomaly can be detected for low torque loads. Besides that, the faulty phase classification provides almost 100 % accuracy for load torques greater than 5 Nm and rotational speeds ranging from 500 to 4400 rpm. Although the implementation has drawbacks in diagnosis time, it is worth considering due to potential implementation improvements. Also, assuming scenarios where an electric machine operates within a moving vehicle, immediate mitigation may not be possible, allowing more time for diagnosis. In future work, we plan to include a severity factor of the detected and classified anomaly to extract additional information to identify a connection to the number of shorted turns.

References

1 C. Andersson, J. Åkesson, and C Führer. Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface. technical report in mathematical sciences. In *Technical Report in Mathematical Sciences*, volume LUTFNA-5008-2016. Centre for Mathematical Sciences, Lund University, 2016.

2 Hüseyin Tayyer Canseven and Abdurrahman Ünsal. Performance improvement of fault-tolerant control for dual three-phase pmsm drives under inter-turn short circuit faults. In *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–5, 2021. doi:10.1109/IECON48115.2021.9589578.

3 Seungdeog Choi, Moinul Shahidul Haque, Md Tawhid Bin Tarek, Vamsi Mulpuri, Yao Duan, Sanjoy Das, Vijay Garg, Dan M. Ionel, M. Abul Masrur, Behrooz Mirafzal, and Hamid A. Toliyat. Fault diagnosis techniques for permanent magnet ac machine and drives—a review of current state of the art. *IEEE Transactions on Transportation Electrification*, 4(2):444–463, 2018. doi:10.1109/TTE.2018.2819627.

4 Cheng Fan, Meiling Chen, Xinghua Wang, Jiayuan Wang, and Bufu Huang. A review on data preprocessing techniques toward efficient and reliable knowledge discovery from building operational data. *Frontiers in Energy Research*, 9, 2021. doi:10.3389/fenrg.2021.652801.

5 Felipe L. Gewers, Gustavo R. Ferreira, Henrique F. De Arruda, Filipi N. Silva, Cesar H. Comin, Diego R. Amancio, and Luciano Da F. Costa. Principal component analysis: A natural approach to data exploration. *ACM Comput. Surv.*, 54(4), May 2021. doi:10.1145/3447755.

6 The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022. URL: https://www.mathworks.com.

7 David Kaufmann. Simulation-Based Diagnosis Method for Dual Three-Phase E-Motor. Software, version 1.0. (visited on 2024-11-12). URL: https://doi.org/10.5281/zenodo.14026570.

8 David Kaufmann, Iulia Nica, and Franz Wotawa. Intelligent agents diagnostics - enhancing cyber-physical systems with self-diagnostic capabilities. *Advanced Intelligent Systems*, 3(5):2000218, May 2021. doi:10.1002/AISY.202000218.

9 David Kaufmann and Franz Wotawa. A framework for integrating automated diagnosis into simulation. In *Industrial Artificial Intelligence Technologies and Applications*, pages 113–127. River Publishers, June 2022. doi:10.13052/rp-9788770227902.

10 David Kaufmann and Franz Wotawa. Data preprocessing for utilizing simulation models for ml-based diagnosis. *IFAC-PapersOnLine*, 58(4):19–24, 2024. 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024. doi:10.1016/j.ifacol.2024.07.187.

11 Matus Kozovsky. Measurement of interturn short-circuits emulation on dual three-phase PMS motor, October 2024. doi:10.5281/zenodo.13889418.

12 Matus Kozovsky, Ludek Buchta, and Petr Blaha. Compensation methods of interturn short-circuit faults in dual three-phase pmsm. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 4833–4838, 2020. doi:10.1109/IECON43393.2020.9254734.

13 Matus Kozovsky, Ludek Buchta, and Petr Blaha. Interturn short circuit modelling in dual three-phase pmsm. In *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6, 2022. doi:10.1109/IECON49645.2022.9968364.

14 Matus Kozovsky, Ludek Buchta, and Petr Blaha. Implementation of ann for pmsm interturn short-circuit detection in the embedded system. In *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6, 2023. doi:10.1109/IECON51785.2023.10312642.

15 Matúš Kozovský. *Modeling and Control of AC Electric Drives During Fault Conditions*. PhD thesis, Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation, 2020.

16 Bo Liu, Yuan-rui Zhang, Xue-lian Cao, Yu Liu, Bin Gu, and Tie-xin Wang. A survey of model-driven techniques and tools for cyber-physical systems. *Frontiers of Information Technology & Electronic Engineering*, 21(11):1567–1590, 2020. doi:10.1631/FITEE.2000311.

17 Modelica Association. Functional mock-up interface, 2021. URL: https://fmi-standard.org/.

18 Teresa Orlowska-Kowalska, Marcin Wolkiewicz, Przemyslaw Pietrzak, Maciej Skowron, Pawel Ewert, Grzegorz Tarchala, Mateusz Krzysztofiak, and Czeslaw T. Kowalski. Fault diagnosis and fault-tolerant control of pmsm drives–state of the art and future challenges. *IEEE Access*, 10:59979–60024, 2022. doi:10.1109/ACCESS.2022.3180153.

19 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

20 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

21 Fatemeh Mohammadi Shakiba, S. Mohsen Azizi, Mengchu Zhou, and Abdullah Abusorrah. Application of machine learning methods in fault detection and classification of power transmission lines: a survey. *Artificial Intelligence Review*, 56(7):5799–5836, 2023. doi:10.1007/s10462-022-10296-0.

22 Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020. doi:10.1016/j.asoc.2019.105524.

23 Cameron Sobie, Carina Freitas, and Mike Nicolai. Simulation-driven machine learning: Bearing fault classification. *Mechanical Systems and Signal Processing*, 99:403–419, 2018. doi:10.1016/j.ymssp.2017.06.025.

24 I. Stančin and A. Jović. An overview and comparison of free python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 977–982, 2019. doi:10.23919/MIPRO.2019.8757088.

25 Peter Struss. Fundamentals of model-based diagnosis of dynamic systems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 480–485. Morgan Kaufmann, 1997.

26 Romain Tavenard. An introduction to dynamic time warping. https://rtavenar.github.io/blog/dtw.html, 2021.

27 Hasan Torabi, Seyedeh Leili Mirtaheri, and Sergio Greco. Practical autoencoder based anomaly detection by using vector reconstruction error. *Cybersecurity*, 6(1):1, 2023. doi:10.1186/s42400-022-00001-0.

28 Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016. RoLoD: Robust Local Descriptors for Computer Vision 2014. doi:10.1016/j.neucom.2015.08.104.

29 Meert Wannes, Hendrickx Kilian, Van Craenendonck Toon, Robberechts Pieter, Blockeel Hendrik, and Davis Jesse. Dtaidistance, October 2022. doi:10.5281/zenodo.7158824.

Achieving Complete Structural Test Coverage in Embedded Systems Using Trace-Based Monitoring

Alexander Weiss[1]
Accemic Technologies GmbH, Kiefersfelden, Germany

Albert Schulz
Accemic Technologies GmbH, Kiefersfelden, Germany

Martin Heininger
Heicon Global Engineering GmbH, Schwendi, Germany

Martin Sachenbacher
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Martin Leucker
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

Abstract

This paper presents a systematic approach to achieving, in a well-defined sense, 100% structural test coverage for large embedded software projects. In embedded systems, high code coverage is a critical part of the testing process to ensure that the system works correctly. Measuring code coverage provides insight into the effectiveness of the testing process, the quality of the software, and can help identify untested or partially tested areas of the code. Traditionally, coverage is often measured when unit tests are executed. The proposed approach instead uses integration tests as the starting point for determining test completeness. Measuring code coverage at the integration test level in embedded systems can be challenging due to the limitations of software instrumentation (additional memory requirements and additional CPU load). To overcome these limitations, embedded trace technology is used to measure code coverage continuously and non-intrusively. The use of these techniques will help to increase the reliability of embedded software and reduce the likelihood of missed integration tests, missed high-level requirements, and undetected software defects.

2012 ACM Subject Classification Hardware → Reconfigurable logic applications; Hardware → Coverage metrics; Software and its engineering → Software testing and debugging

Keywords and phrases structural tests, integration tests, code coverage, embedded trace

Digital Object Identifier 10.4230/OASIcs.DX.2024.19

Category Short Paper

Funding TRISTAN received funding from the EU Key Digital Technologies Joint Undertaking (KDT JU) under project ID 101095947, and the German Federal Ministry of Education and Research (BMBF) under project ID 16MEE0273. Martin Sachenbacher has been partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number 01MC22005C.

1 Introduction

A crucial phase in software development is requirements management, which includes the gathering, analysis, modification, and especially the refinement of requirements. Requirements management is a complex task, a distinct research field in computer science, and many projects fail due to unclear and imprecise requirements. Requirements define what the system

[1] corresponding author



© Alexander Weiss, Albert Schulz, Martin Heininger, Martin Sachenbacher, and Martin Leucker;
licensed under Creative Commons License CC-BY 4.0



to be built should do. Therefore, it is essential to check whether the resulting system meets its requirements. A widespread method for verifying this is testing. The goal here is to use appropriate test cases to verify that the developed system meets its requirements.

Another phase in software development is the so-called high-level design of the application, in which the overall system architecture is defined. The high-level design is refined into a low-level design, where individual units, known as "units", and their interactions within the system to be developed are specified. In the V-model, which is a process model describing the chronological sequence of various phases [20, Appendix B], the requirements and design phases are located on the left side of the "V". The individual units are tested and gradually integrated into larger units until the complete system is formed. The idea of the V-model is to thoroughly test each unit and then verify the correctness of component integration. This testing phase is located on the right side of the "V".

As mentioned above, the requirements represent exactly what the system should accomplish. Therefore, it is usual and standard practice to develop corresponding test cases for each individual requirement, which demonstrate in the overall system that the requirements are met. It is possible that a test case may check aspects of various requirements, so an $n : m$ relationship between requirements and test cases is established.

From a maintenance perspective, it is important that the implemented system does not perform more than what is required by the specifications. A well-established approach in practice is to measure how well test cases cover the system's code. If code exists for which no test cases are present, this is a sign that the test suite is incomplete – either requirements have been implemented that are not covered by test cases, or unnecessary functionality has been implemented. Therefore, it is desirable for 100% of the code to be covered by test cases. Code coverage is defined as the degree (in percent) to which the source code of a program is executed/accessed when a particular test suite is run. Coverage can be measured using different criteria such as statement coverage, branch coverage, condition coverage, MCDC coverage, etc. These criteria, simply put, require varying levels of thoroughness and should be used depending on the criticality (risk) of the system being tested. High code coverage of the test suite reduces the chance that the program contains undetected software defects and is an indication of test completeness. Achieving high code coverage is therefore important for rigorous system testing, and standards often specify which coverage criteria must be met based on the risk of the subsystem.

The requirements and their corresponding test cases must match the granularity of the subsystem being tested. Requirements for the entire system result in test cases for the entire system, known as system tests. Requirements for larger subsystems lead to tests that check the integrated subsystems, called integration tests. Requirements for individual units result in unit tests. Since a fully integrated system and corresponding integration tests check the entire system, integration tests are of particular importance for verifying the whole system. Therefore, conducting integration tests is crucial.

In practice, the focus is however often on testing individual units. These can generally be tested well, i.e., with high coverage and efficiency. It is then assumed that testing the entire system can be done with less effort due to the correctness of the individual units. Consequently, integration tests are performed only to a limited extent. However, especially for embedded systems, the behavior of integrated systems is not fully defined by individual units, as aspects like timing, memory usage, performance, concurrency, etc., only become challenges in the integrated system. There are in fact several advantages of measuring test completeness at the integration test level compared to the unit test level, particularly in the domain of embedded and safety-critical software. For example, integration tests replicate

real-world scenarios by combining multiple software components. This approach provides a more accurate representation of how the software will function in its intended environment, which is crucial for safety-critical systems. Moreover, integration tests cover interactions and interfaces among different software modules, ensuring that the entire system functions cohesively. This is especially important in embedded systems where multiple components must work together seamlessly. These tests help identify potential problems that may not surface in isolated unit tests, contributing to higher defect detection and risk mitigation. A vivid example of this is the Ariane V spacecraft, which was destroyed during its maiden flight after a deviation from the required flight path was detected. The cause was identified as a well-tested function that had worked successfully in the predecessor Ariane IV. Due to the changes in the overall system, however, this functionality was no longer adequate, leading to the catastrophic failure. This is another reason why integration tests should receive greater importance in practice than they currently do. Overall, integration tests play a critical role in validating safety-critical software, ensuring regulatory compliance, and evaluating complex use cases while providing insight into system performance.

Testing a fully integrated system and measuring coverage on an embedded system, however, is a major challenge. Usually this requires a test system on the target platform and software instrumentation, i.e. additional code that is executed alongside the actual system on the target platform and provides the necessary information for analysis; an example is gcov [11]. However, for embedded systems integration tests often cannot be conducted because the resources of the target platform are only sufficient for the actual application and not for both the application and the test system.

The instrumentation and simultaneous execution of the test system and the actual application distorts the behavior of the application to some extent, as additional code (even if it only outputs variables) affects the behavior of the system. This so-called *probe effect* is particularly problematic when analyzing timing behavior and concurrency-related errors such as race conditions on multi-core platforms. It will potentially cause resource-critical requirements (e.g., timing, memory) to be mistakenly marked as fulfilled or unfulfilled by test cases. The modification thus limits the measurement of code coverage during the execution of integration tests, restricting the meaningful measurement of code coverage to the execution of unit tests only.

In this paper, a non-invasive monitoring method for application execution is presented, which is based on recent technological advances [18, 1] that allow the monitoring and analysis part to essentially run on additional specialized hardware (FPGAs), so that the application itself is not affected. This ensures that integration tests can be conducted without running a test system on the target hardware, and the test results apply to the actual runtime, allowing code coverage to be measured during the execution of integration tests without additional instrumentation. We build on this technical solution for continuous and non-intrusive monitoring of processors to propose a systematic approach for determining and achieving test completeness in terms of code coverage at the integration level. In practice, achieving complete coverage of code, even with the simplest coverage criterion, statement coverage (where every single line of code must be executed once), is usually not feasible for several reasons laid out in the next section. Therefore, achieving 100% code coverage requires either an appropriate test case linked to the requirement or a rationale explaining why certain coverage criteria cannot be met. In other words, coverage is measured as the sum of coverage by test cases and coverage by justification, making 100% coverage achievable. Combined with running tests on separate hardware, this ensures an adequate verification of requirements across the entire system. We claim that the approach brings us significantly

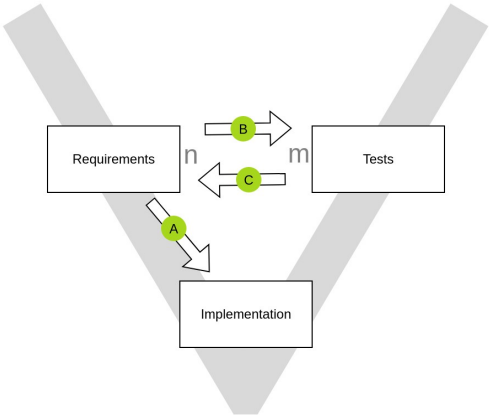


Figure 1 The triad of requirements, implementation, and tests.

closer to the maxim, *what you run, and run what you test* (in the aerospace domain, *test like you fly, and fly like you test* [13]). The actual system is tested, without any additional annotations, and it can later run unchanged in the production environment.

The presentation is structured as follows: in the next section, we review the well-known software development process based on requirements, code implementation, and testing. We then describe our systematic approach for identifying and closing gaps between requirements, tests, and the implementation. The third section provides an introduction to the technical implementation, including the requirements for the observation of integration tests and an introduction to the capture and processing of embedded traces.

2 A Systematic Approach for Reaching 100% Code Coverage

2.1 Software development process

Figure 1 shows the classic V-model, consisting of requirements, implementation, and their associated tests.

In software engineering, the transition from requirements to implementation (A) is complex. Engineers write code to implement requirements, but it is important to note that this process is not one-to-one. To verify the correctness of the implementation, appropriate tests are developed and executed based on the requirements.

This practice, known as *requirement test coverage*, ensures thorough software validation. Each requirement requires at least one test (usually several in practice) to systematically validate, improve quality, and meet industry standards such as DO-178C (Software Considerations in Airborne Systems and Equipment Certification) [4], ISO26262 (Functional Safety of Road Vehicles) [8], EN50128 (Railway applications) [5] and IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems) [3].

There is an *n:m* relationship between requirements and tests, where for each requirement there must be at least one corresponding test (B), and likewise for each test there must be at least one corresponding requirement (C). In the industrial area, established application lifecycle management tools such as Polarion [14] or Codebeamer [9] provide the management of requirements and tests, but there are also new solutions such as the AI-supported MappingSpace [12]. The first step (A) to (C) serves as an essential test completeness criterion, but it is not sufficient to guarantee that all implemented code has been tested.

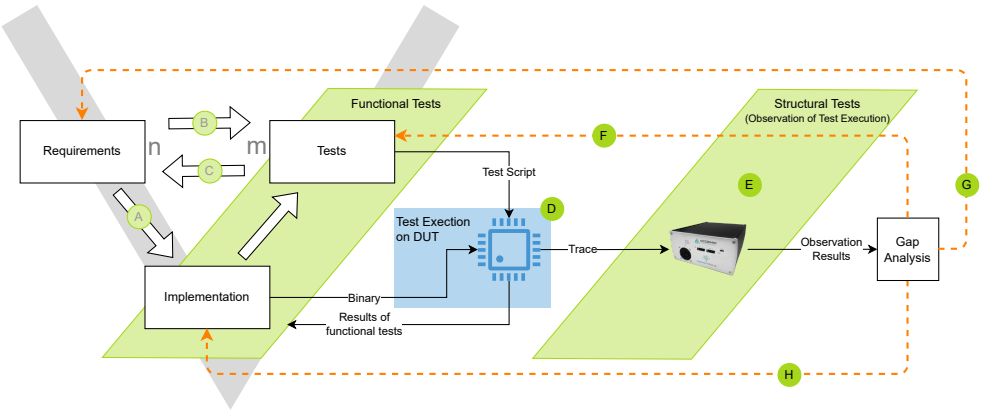


Figure 2 Systematic gap analysis identifies and eliminates gaps between requirements, testing, and implementation.

During the execution of the functional tests (D), the code coverage is measured to determine if all code elements have actually been executed (also called structural tests (E)). Typically, coverage gaps are observed at this stage (code coverage $< 100\%$) and it is necessary to carefully analyze the coverage gaps to identify potential problems or areas for improvement. Figure 2 illustrates the process of test execution, completeness measurement, and gap analysis.

In practice, achieving 100% code coverage is usually not feasible and there are often plausible explanations for coverage gaps, some of which we will outline here:

- Defensive programming: These techniques aim to catch errors and handle specific situations that should not actually occur, leading to code that cannot be reached. An example in the C programming language is the typical pattern `if (p == NULL) {` ..., which is to ensure that `p` is only accessed if it is not a null pointer. If the program is correct, `p` will never be a null pointer, and there will be no test case that makes the condition true.
- Test environment economics: In some cases, it may not be feasible or cost-effective to test certain aspects of the code, such as functional behavior outside of laboratory temperatures or simulating all external failure cases. Especially in cases of low system criticality, it is acceptable if some requirements are not met.
- Safeguards against critical environmental conditions: To ensure the safe operating range of sensors, subsystems, etc., environmental parameters are checked, and undesired or uncertified system usage is blocked by corresponding error routines. A measuring system, for example, might be certified for operation at ambient temperatures, and refuse to operate above that temperature. It may not be safe to test certain aspects of the code such as safeguards against critical environmental conditions, or even though the system could technically be tested in that environment, it is unnecessary as it would not be used as intended.
- Modular nature of the system: From an economic perspective, well-tested units should be reused as much as possible. Units are designed to be used in different contexts, but if only part of the unit's functionality is used in the overall system, the other parts cannot be covered by tests. For instance, consider a controller function that can be set to operate in proportional, differential, or integral modes; if only differential mode is used in the current application, the other modes cannot be tested by integration tests. However,

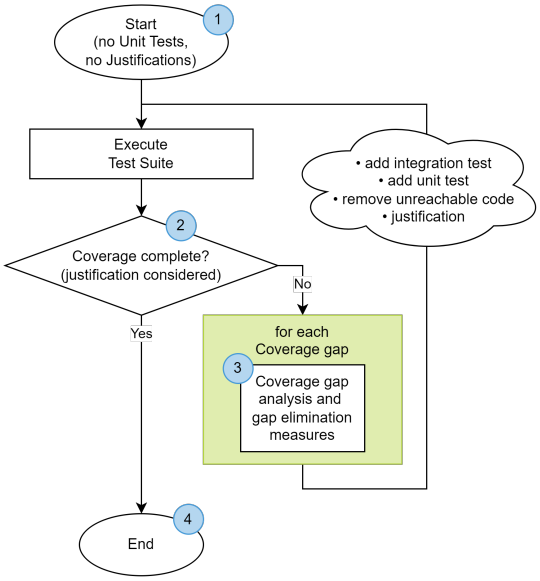


Figure 3 Systematic process for identification and elimination of coverage gaps.

reusable units should have corresponding unit tests to ensure their quality independently of the current application. In this case, unit tests can justify that parts of the code not covered by integration tests have nevertheless been sufficiently tested.

Therefore, achieving 100% code coverage requires either an adequate test case linked to the requirement, or a rationale explaining why certain coverage criteria cannot be met. In other words, coverage is measured as the sum of coverage by test cases and coverage by justification, making 100% coverage achievable.

The measured code coverage of multiple test executions is aggregated to identify the remaining coverage gaps (Figure 4).

If it is determined that there is a plausible explanation for certain coverage gaps, those gaps can be excluded from the measurement. This means that the coverage gaps will not be counted as part of the overall coverage and will not be included in the calculation of the coverage percentage.

However, there must be detailed documentation of the reason for each justification.

To support the exclusion of justified coverage gaps from the measurement, and to identify missing tests (F), missing requirements (G), and unreachable code (H), a systematic process for identifying and excluding these gaps must be implemented. This process is a critical step in the testing process and is designed to ensure that coverage results are accurate.

2.2 Rigorous identification and elimination of coverage gaps

Figure 5 illustrates the systematic process for incrementally identifying and eliminating of coverage gaps. The process typically follows these steps:

(a) Measure the initial test coverage of all integration tests (excluding unit tests) (1). This provides a baseline for the testing process and is used as a starting point for coverage gap analysis.

(b) Execute tests and measure code coverage (2).

(c) Review and analyze each coverage gap (3).



Figure 4 Aggregation of multiple coverage results.

(d) If the gap is justifiable, exclude it from the measurement. If the gap is not justifiable, run additional tests to increase coverage and close the gap.

(e) Repeat the process until all coverage gaps have been reviewed and analyzed, and the sum of covered and excluded code sections equals 100% of the code (4).

It is important to emphasize that this work should not be done too early in the development cycle, as it can be very costly. Instead, it is preferable to establish a stable software base and achieve full requirements test coverage before starting this work.

Each coverage gap must be carefully reviewed and analyzed to determine the cause of the gap (see label 3A in Figure 5). If it is determined that the gap is caused by a missing integration test (3B), an additional integration test (3E) can be added to increase coverage and close the gap. Before adding the additional integration test, it is important to check if there is also a missing requirement (3C) that needs to be added. If there is a missing requirement, it will also be added to the project (3D).

If the coverage gap analysis (3A) does not identify any missing integration tests, then the presence of unreachable code (3F) must be checked. Unreachable code is code that is not executed or used during the testing process and must be removed from the code base to improve the overall quality and maintainability of the code (3L).

If there is no unreachable code, consideration should be given to whether the gap can be filled by a unit test (3G). Code may not be testable by an integration test, requiring a unit test instead, under conditions such as (but not limited to) specific physical constraints. For example, Figure 6 illustrates a situation where the operating range of a sensor prevents it from being run in an integration test, but this range can still be examined by a unit test.

Adding a unit test (3H) to the test suite is only allowed in justified circumstances and must be thoroughly documented (3J). Otherwise, there's a risk that unit tests will mask the absence of integration tests - a common problem in traditional unit test-focused coverage evaluations. It is critical to distinguish between integration and unit tests to avoid masking integration-level coverage gaps with unit tests.

However, the initial exclusion of unit tests and their inclusion only in justified cases should not lead to a general undervaluation of unit tests. They remain essential, especially during development.

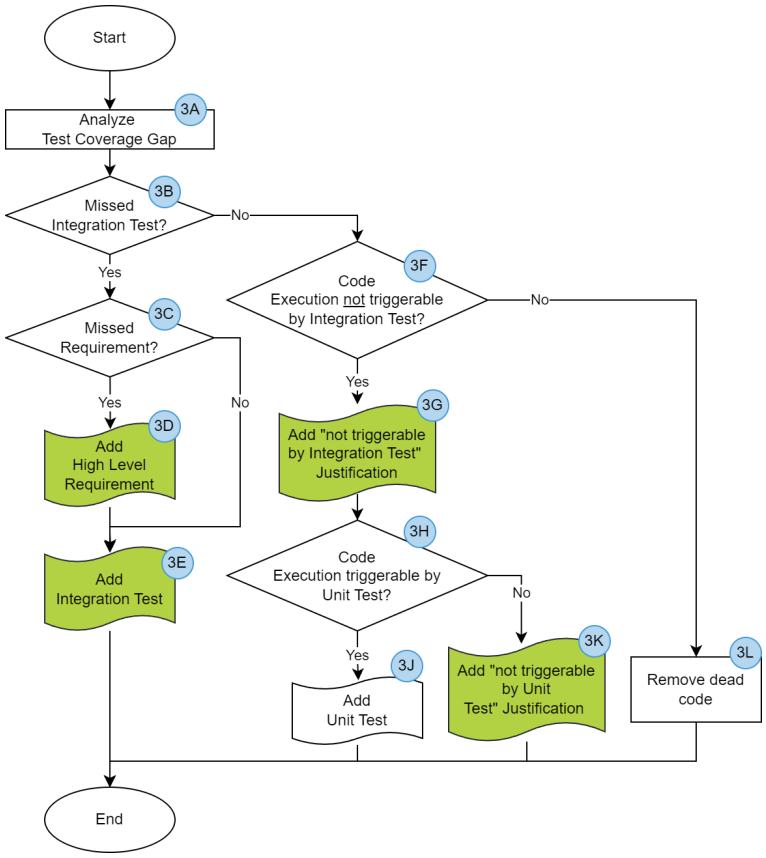


Figure 5 The process of identifying and eliminating potential coverage gaps. The unit tests added here (with a detailed explanation of why they are needed at this point) are added to the test base, which initially contains no unit tests. The green boxes highlight the potential benefits of using this approach compared to proving full code coverage based on unit testing alone. The highlighted blocks represent the improvement by integration test coverage achieved with the proposed method.

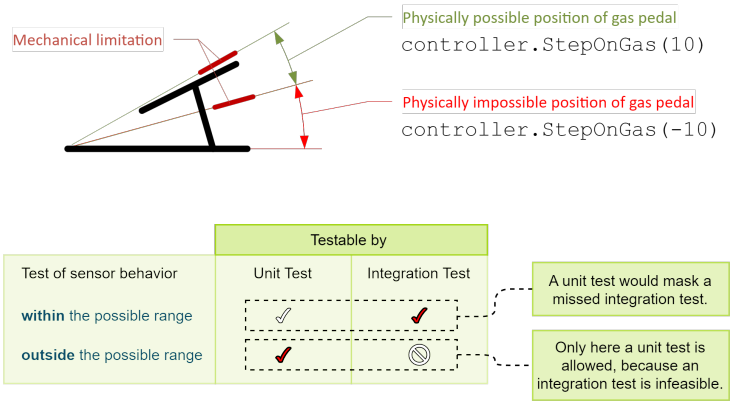


Figure 6 Example of a case where a unit test is acceptable because the test cannot be performed as an integration test. The throttle can only be moved within a mechanically defined range; tests with irregular sensor values must be performed as unit tests.

	Test Level	
Observation Requirements	***Unit Tests***	***System and Integration Tests***
Long Observation	Not required	Required
No (or minimal) change in Timing Behavior		
No (or minimal) change in Memory Footprint		

Figure 7 Observation Requirements for System, Integration, and Unit Testing.

If the gap cannot be covered by a unit test, a justification for the coverage gap is documented (3K). This documentation provides a record of the gap and the reasons why it was considered justifiable and excluded from measurement.

The goal of complete structural test coverage at the integration test level can be achieved by systematically adding missing tests, removing code sections that cannot be tested and carefully documenting the justification of code sections that are necessary but cannot be tested.

In particular, measuring code coverage is a mandatory requirement for certification according to standards such as DO-178C (Software Considerations in Airborne Systems and Equipment Certification) [4], ISO26262 (Functional Safety of Road Vehicles) [8], EN50128 (Railway applications) [5] and IEC 61508 (Functional safety of electrical/electronic/ programmable electronic safety-related systems) [3]. However, these standards leave open the level at which code coverage must be verified.

3 Technical Solution

3.1 Requirements

The process described seems intuitive and obvious. The question arises as to why structural coverage has mostly been determined at the unit test level and why the consistent step to higher test levels has not yet been taken for embedded systems.

The answer can be found by analyzing the requirements for observing an embedded processor (Figure 7). These are low for unit tests; only the control flow in an isolated code element needs to be examined. The method of automated software instrumentation has been used successfully for years. All relevant places in the software are provided with additional code that logs the execution of that place. This typically involves reading an execution counter from memory, incrementing it, and writing it back. The associated bus operations are non-deterministic on multiple processors.

However, software instrumentation negatively impacts integration testing: the additional code required for instrumentation consumes memory and processing time, both of which are limited resources in embedded systems. If there is not enough memory for the instrumentation, the instrumented application will not run. Even if this is still the case, the changed runtime behavior of the instrumented software version represents a significant limitation in the validity of the tests performed.

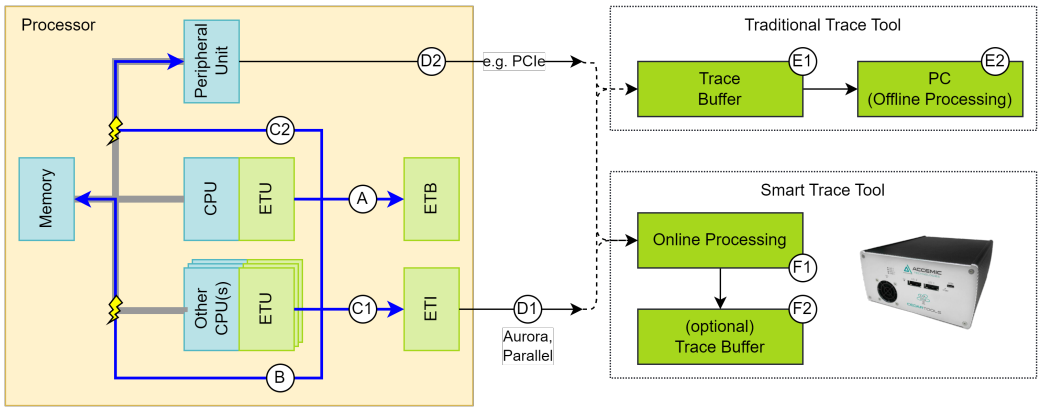


Figure 8 Embedded trace elements.

3.2 Embedded Trace

A suitable solution is to use the embedded trace [19] resources available on almost every processor. This technology is implemented through specific hardware structures that are tightly coupled to the CPU(s) and are used to capture data about the execution of code. The major advantage of using embedded trace is that the embedded trace unit (ETU) does not affect the CPU when monitoring the program flow, i.e. from the application's point of view it is not possible to tell whether the ETU is active or not. Figure 9 provides an overview of the individual elements and the trace data flow of a processor equipped with embedded trace.

There are several options for transferring the trace data generated by the ETU to an external tool. The trace data can be temporarily stored in the processor in a special embedded trace buffer (ETB) (A) or in the system memory (B). Trace data can also be output via dedicated embedded trace interfaces (ETI) (C1) or system interfaces (C2) to avoid the limited observation time caused by the memory limitations.

The direct output of the trace data via ETI (C1) is technically the most elegant, since it does not interfere with the processor. However, implementing an interface with the required high bandwidth is expensive, so some processor designs compromise by storing the trace data in the main memory (B) or outputting it via a system interface (C2). However, this is also done at the expense of the desired non-intrusiveness - the required bus operations interfere with the application (indicated by the yellow flashing symbols).

As shown in Figure 8, trace data is output directly using dedicated trace interfaces (parallel [6] or high-speed serial like Aurora [2]) (D1) or fast system interfaces (usually PCIe, sometimes USB or Ethernet) (D2).

This trace data is then received by an external trace tool. There are simple trace buffers and advanced smart trace tools.

Trace buffers (E1) typically contain several GB of memory where the received trace data is temporarily stored before the decoding and further processing in a PC (E2). The downside is that the size of the trace buffer is limited, which limits the amount of time that can be observed. This conflicts with the requirement of an integration test to be able to observe and analyze a system for as long as possible.

This problem is solved by a new generation of smart trace tools (F1) that perform live reconstruction of the control flow and compute online all the information needed to determine the code coverage of the observed integration tests (Figure 9). In principle, such a system

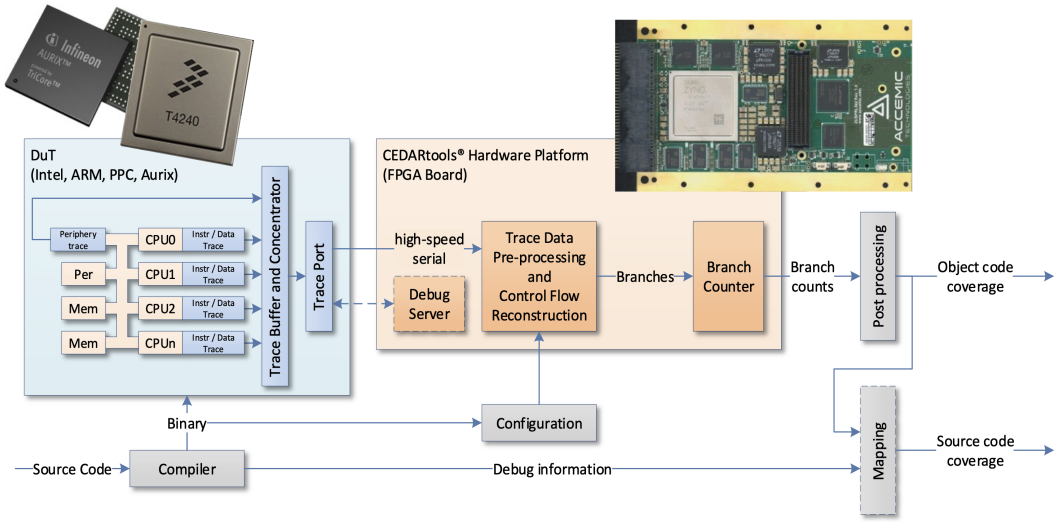


Figure 9 Embedded Trace-based Coverage Generation.

does not require a trace buffer, since the incoming trace data is processed immediately and continuously online. However, the system also has a trace buffer with a capacity of several GB (F2) for evidentiary purposes and other potential applications such as test impact analysis or hybrid WCET measurement. Since the control flow of the target system has been reconstructed in the system, the existing trace buffer can store the relevant trace snippets in a finely segmented manner more efficiently than with simple trace buffers.

All trace-based tools determine the control flow at object code level. Using the DWARF debug information [7], the object code coverage can be transformed into the source code coverage.

This transformation is not trivial, because an optimizing compiler can change the control flow. This is done by removing, adding (e.g. stack protection, vectorization), or rearranging code elements. These compiler-induced changes to the control flow should also be systematically tested. So far, these gaps have not been addressed in the applicable standards. The new state of the art, which makes this additional coverage analysis possible, must be brought to the attention of experts and taken into account in binding standards.

In addition, especially for safety-critical applications, it must be considered that a compiler may also be incorrectly implemented [10].

Trace-based code coverage and thus the method presented here works well for simple coverage levels such as call function coverage, source line coverage, or statement coverage. More sophisticated coverage metrics, such as branch coverage or MC/DC [17], require control-flow preserving compilation and reliable debug information. The authors are not currently aware of any viable solutions in this area, but intensive development work is underway to achieve a broadly applicable solution.

The previous discussion has shown how important embedded trace is to achieving the highest possible software quality for embedded systems. Established processors offer complex embedded trace solutions. A closer analysis of various use cases (such as the one presented here) reveals some gaps and functional limitations in the established embedded trace solutions. As part of the TRISTAN project [16], the authors are working on the development of an embedded trace IP (open source and commercial version) for the open standard instruction set architecture RISC-V [15] that comprehensively addresses the use case presented here.

4 Summary

We showed that by systematically detecting and eliminating coverage gaps, and missing integration and unit tests, it is possible to identify and eliminate missing requirements and unreachable code. A key enabler of this approach is smart trace tools that provide continuous and non-intrusive monitoring of embedded processors, ensuring the integrity of the program flow during execution. These technical developments now also allow code coverage to be measured during the execution of integration tests. The resulting approach thus brings us closer to fulfilling the important principle *test what you run, and run what you test.*

References

1 CEDARtools. Available: `https://accemic.com/cedartools/`.
2 Aurora Protocol Specification, 2007. Technical Report, Xilinx Inc.
3 IEC 61508:2010 Functional safety of electrical/ electronic/programmable electronic safety-related systems, 2010.
4 DO-178C, Software Considerations in Airborne Systems and Equipment Certification, December 2011.
5 EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011.
6 MIPI PTI v2.0 Specification for Parallel Trace Interface, May 2011. Technical Report, MIPI Alliance, Inc.
7 DWARF Debugging Information Format Version 5, February 2017. Available: `http://www.dwarfstd.org`.
8 ISO 26262:2018. Road vehicles – Functional safety, 2018.
9 Codebeamer. Available: `https://www.ptc.com/en/products/codebeamer`.
10 GCC Bug-tracking System. Available: `https://gcc.gnu.org/bugzilla/`.
11 Gcov. Available: `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`.
12 MappingSpace. Available: `https://www.ytdevops.com/home`.
13 NASA lessons. Available: `https://llis.nasa.gov/lesson/1196`.
14 Polarion. Available: `https://plm.sw.siemens.com/en-US/polarion/`.
15 RISC-V. Available: `https://riscv.org/`.
16 The TRISTAN Project. Available: `https://tristan-project.eu/`.
17 K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. A practical tutorial on modified condition/decision coverage. Technical report, NASA Langley Technical Report Server, 2001.
18 Martin Leucker, Martin Sachenbacher, Simon Wegener, Alexander Weiss, and Albert Schulz. Non-intrusive, continuous trace-based monitoring and its applications for system correctness, safety, and resilience. In *Workshop on Principles of Diagnosis (DX'23)*, 2023.
19 T. B. Preußer, S. Gautham, A. D. Rajagopala, C. R. Elks, and A. Weiss. Everything you always wanted to know about embedded trace. *Computer*, 55(2):34–43, February 2022. `doi:10.1109/MC.2021.3098965`.
20 Tim Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. *Model-Based System Architecture*. Wiley Series in Systems Engineering and Management. John Wiley & Sons, 2nd edition, 2022.

Data-Driven Diagnosis of Electrified Vehicles: Results from a Structured Literature Review

Stan Muñoz Gutiérrez
Institute of Software Technology, TU Graz, Austria

Adil Mukhtar
Institute of Software Technology, TU Graz, Austria

Franz Wotawa[1]
Institute of Software Technology, TU Graz, Austria

Abstract

Traditional onboard vehicle diagnostics are rapidly evolving concomitant to the rise of electrified powertrains, digital transformation, and intelligent technologies for advanced system management. The big data now available in modern vehicles offers unprecedented opportunities for condition monitoring and prognosis, but also presents challenges in scaling and integrating multimodal sensor data across components with varying timescale dynamics. Machine learning techniques have proven particularly effective in implementing diagnostic functions within electrified vehicle powertrains. This study systematically reviews intelligent, data-driven techniques for health monitoring and prognosis of electrified powertrains. We categorize existing research based on diagnostic functions and machine learning methods, with a focus on approaches that do not require prior knowledge of faulty operational states. Our findings indicate that deep learning methods are state-of-the-art across several diagnostic functions, fault modes, system levels, and multimodal sensor integration.

2012 ACM Subject Classification Computing methodologies → Machine learning algorithms; Hardware → Safety critical systems; Hardware → Failure recovery, maintenance and self-repair

Keywords and phrases Diagnostic functions, Machine Learning, Powertrain, Electrified vehicles

Digital Object Identifier 10.4230/OASIcs.DX.2024.20

Category Short Paper

Supplementary Material *Audiovisual (Poster)*: `https://zenodo.org/records/14065416`

Funding This study was conducted within the framework of the project ARCHIMEDES, receiving funding from the Key Digital Technologies Joint Undertaking (KDT JU) - the Public-Private Partnership for research, development, and innovation under Horizon Europe – and National Authorities under Grant Agreement No 101112295 funded by the European Union and the FFG under Grant Agreement No FO999899377.

1 Introduction

The digital transformation within the transport system is rapidly changing the landscape of automotive engineering processes for developing, operating, and managing the end-of-life of vehicles. One of the primary motivations for the electrification of powertrains is to reduce their ecological footprint by increasing energy transformation efficiency and moving towards zero CO_2 emissions [22]. At the same time, this transformation aims to improve transportation safety. Active safety systems (ASSs) engage transiently in the event of detection of internal or external hazards, they react timely providing warnings, adjusting the parameters or control of the vehicle, yet without assuming the responsibility and role of the

[1] corresponding author


© Stan Muñoz Gutiérrez, Adil Mukhtar, and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



human driver [44]. On the other hand, Advanced Driver Assistance Systems (ADASs) are expected to engage continuously as they increase the level of *driving automation* defined in the SAE J3016 standard [43]. At level 5, vehicle decision-making is fully automated, requiring of more comprehensive health awareness capabilities. Vehicle powertrain electrification brings unprecedented possibilities regarding vehicle control, eco-driving, real-time health monitoring, and predictive maintenance. Data transmission and storage are scaling rapidly in speed and bandwidth to accommodate the volume of information generated by the many digital sensors available in the intra-vehicle networks connecting the different components of an electrified powertrain (EP). The traditional component-based onboard diagnostics, prescribed in the *On-Board Diagnostics* (OBD-II) and the ISO 14229-1 *Unified Diagnostic Services* (UDS) standards, are expected to transition to hierarchical intelligent health management systems.

Diagnostic solutions can be developed using different conceptual frameworks. In the case of motors, but equally appropriate for other components of the EP, Chen et al. [5] classifies the approaches into *model-based*, *signal-based*, and *data-driven*. Within model-based methods, the classification can be refined into analytical mathematical models, magnetic equivalent circuits (MECs), and digital simulation models [2]. The choice of the kind of model is driven by accuracy, computational complexity, and in many cases the type of faults to consider. As for signal-based diagnosis techniques, they are mainly concerned with characterizing the different fault signatures. They rely on feature extraction techniques that operate in the time, spectral, or time-frequency domains. Because of its sound theoretical foundations, these techniques are often incorporated in data-driven approaches. Data-driven diagnosis is primarily enabled by machine learning (ML) and artificial intelligence (AI) models.

In this paper, we review the current research landscape of data-driven Intelligent Diagnostic Systems (IDSs) for EPs and present the first installment of a comprehensive literature survey currently in the making. Section 2 details the methodology of our research, including research questions (Section 2.1), search criteria (Section 2.2), inclusion criteria (Section 2.3), and our data processing approach (Section 2.4). Bibliometric facts are presented in Section 3. In Section 4, we discuss the present answers to the research questions. Section 5 interprets and discusses our findings. Finally, Section 6 offers some concluding remarks.

2 Methodology

The methodology of this study is summarized in Figure 1. We start by formulating our research questions. Then we define a search string to query pertinent scientific databases. Various inclusion criteria were defined to narrow down the relevant articles, resulting in a subset of articles that undergo a pre-screening process. This curation step ensures that the selected articles align with the predefined scope and objectives. Finally, we conduct a manual review of the articles to collect our insights and findings regarding various DFs, components and machine learning methods.

We center our study on the Motor Systems (MSs) of EPs. Attending to the fact that healthy behavior data is easier to obtain than faulty behavior data, we also focus on unsupervised machine-learning methods, suitable to work without knowledge nor data from faulty behavior of the system. We aim to shed light on how these methods can deliver effective and reliable EP IDSs. In the following, we provide details about the research questions, search schema, acceptance criteria, and data processing pipeline.

[2] A detailed account of the different approaches and classifications found in the literature can be found in Usman et al. [53]. Usman's taxonomy refers to these classes as electrical equivalent circuit, magnetic equivalent circuit, and numerical methods.

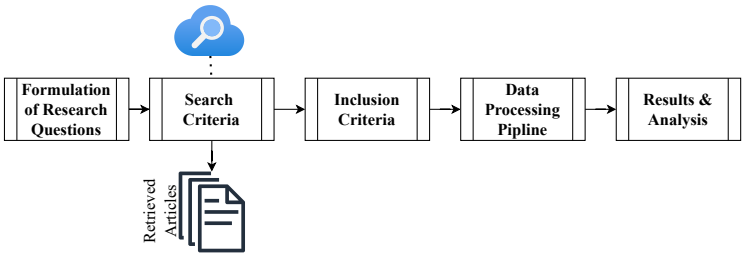


Figure 1 Survey methodology overview.

2.1 Research Questions

1. What are the DFs covered in the research?
2. What components of the EP are considered?
3. What anomalies or faults were included in the study?
4. What are the AI techniques applied to implement the DFs?
5. To what extent is it feasible to develop DFs implemented with unsupervised ML techniques that rely exclusively on unlabeled data corresponding with the healthy behavior of the system?

2.2 Search Criteria

2.2.1 Scientific Portals

Searches performed in: (1) SAE Mobilus, (2) Scopus, (3) IEEE Xplore, (4) TU Graz Library Search, (5) Wiley Online Library, (6) ProQuest, (7) ScienceDirect, and (8) Web of Science.

2.2.2 Search Query

Logic based: “electric vehicle” AND “motor” AND (“machine learning” OR “artificial intelligence”) AND (“fault detection” OR “anomaly detection” OR “diagnosis” OR “condition monitoring” OR “predictive maintenance”).

2.3 Inclusion and Exclusion Criteria

Papers written in languages other than English, articles to which our institution has no access, empty, incomplete, or untraceable records, retracted articles, papers older than 2018, only abstracts, and all non-peer-review publications were excluded from the survey. All battery-subsystem-only, software-only, and cybersecurity-related papers were considered out of scope for this review. The curation step is explained in Section 2.3.1.

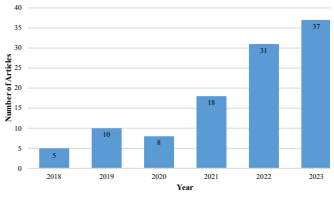
2.3.1 Prescreening Criteria

1. At least one of the following: fault detection, anomaly detection, diagnosis, condition monitoring, or predictive maintenance.
2. At least one of the following: DC link, motor, inverter, and reducer.
3. At least one intelligent data-driven approach.
4. At least one research question is addressed.
5. literature review with at least one work satisfying one or more of the previous criteria.

Selected works either satisfy criteria 1, 2, 3, and 4 simultaneously or they satisfy 5.

Science Portal	Hits	Included (✓)		Excluded (✗)	
		Research	Review	Access	No access
SAE Mobilus	42	1	1	39	1
Scopus	38	16	4	16	2
IEEE Xplore	17	13	0	4	0
TU Graz Library Search	31	15	5	11	0
Wiley Online Library	46	4	2	40	0
ScienceDirect	110	8	1	101	0
ProQuest	506	47	12	447	0
Web of Science	12	8	2	1	1
Total	804	112	27	659	4
Unique	754	86	23	642	3

(a) Summary of results of the prescreening process.



(b) Yearly published articles (RC).

Figure 2 Bibliometric results.

2.3.2 Collections

After the prescreening procedure, the selected works are identified as the *Reference Collection* (RC). In addition, by inspecting each work within the RC we identify additional papers that fulfill our acceptance criteria. These works are referred to as *Extended Collection* (EC). Finally, references preceding 2018 that are relevant constitute the *Baseline Collection* (BC).

2.4 Data Processing

A total of 804 references (754 unique) were prescreened independently using the criteria described in Section 2.3.1. We classified the papers as original research papers or as literature reviews. Being a partial installment, this paper coverage is restricted to 15 IEEE Xplore papers and all publications within the RC between 2018, and 2020, and parts of 2021.

3 Bibliometric Facts

All bibliometric facts are based on the RC exclusively. Table 2a summarizes the classification obtained after the screening procedure. ProQuest portal contributed the highest volume of accepted papers: 42 research papers and 12 review papers (59 in total, or 54% of the final collection), SAE Mobilus the lowest by volume, 1 research paper and 1 review paper (2 in total, or 1.83% of the final collection). On the other hand, the precision of the search engines differs significantly. IEEE Xplore provided only a moderate number of hits (17), yet the prescreening process resulted in an acceptance rate of 76.4%. In contrast, the accuracy of ProQuest was only 11.66%. Figure 2b illustrates the yearly distribution of selected published articles, revealing a growing acceptance of data-driven techniques.

4 Results

What are the DFs covered in the research?

Table 1 summarizes the classification of research papers according to different diagnostic functions. It can be seen that most research items focus on the FD and Diag categories. We classify a work as Diag if at least two different types of faults are included.

Table 1 Publications according to diagnostic function.

Code	Diagnostic function	RC	EC	BC
ADiag	active diagnosis	[27] [38]		
CM	condition monitoring	[30]		
Diag	diagnosis	[1] [34] [35] [38] [49] [50] [52] [55] [61] [63] [64] [65]	[16] [19] [28] [36] [37] [51] [56]	[6] [8] [23] [32] [47] [62]
DTBHM	digital-twin-based health monitoring	[54]		
DTBP	digital-twin-based prognostics	[54]		
FD	fault detection	[2] [4] [13] [17] [29] [31] [33] [40] [48] [57] [58] [59] [60]	[18] [51]	[6] [10] [11] [20] [21] [24] [32] [39] [41] [42] [45] [46] [47] [62] [66]
FE	fault excitation	[27]		
FI	fault isolation	[27]		
FL	fault localization	[2]	[18]	[20] [39] [41] [42] [46]
RA	risk assessment	[25]		
SA	severity assessment	[4] [13] [17] [48] [63]	[9] [16] [28] [56]	[6] [10] [11] [20] [39] [42] [45] [46] [66]
UAD	(unidentified) anomaly detection	[15]		

Table 2 Publications according to EP component.

Code	Component	RC	EC	BC
Agnos	agnostic	[15] [25] [31]		
Inv	inverter	[2] [4] [40] [48]		
HVDCL	high voltage DC line	[57] [58] [59]		
GB	gearbox	[65]	[37]	
Mot	motor	[1] [7] [12] [13] [14] [17] [30] [33] [34] [35] [49] [50] [52] [54] [55] [60] [61] [63] [64]	[9] [16] [18] [19] [28] [36] [37] [51] [56]	[6] [8] [10] [11] [20] [21] [24] [32] [39] [41] [42] [45] [46] [47] [66]
MotS	motor system	[38]		
Sen	sensor	[3] [48] [64]		[62]
PEC	power electronic circuit	[23]		

What components of the EP are considered?

Table 2 shows the coverage of papers per powertrain component. The bias towards motor components is likely a consequence of our search query (see Section 2.2.2).

What anomalies or faults were included in the study?

Table 3 summarizes our findings regarding the types of faults for EPs. There is good coverage of mechanical and electrical faults.

What are the AI techniques applied to implement the DFs?

Table 4 identifies the AI techniques performing intelligent diagnostic functions. It is a common practice to perform multi-technique benchmarks. It can be seen from the frequency of entries alone that among the classic ML techniques, those related to the decision tree

Table 3 Publications according to fault types.

Code	Fault type	RC	EC	BC
AWBF	abrasive wear (BF)		[28]	
BallF	ball fault (BF)	[7] [34] [35] [63] [65]	[19] [37] [51] [56]	[47]
BCD	bearing cage defect			[24]
BiasSF	bias sensor fault	[64]		[62]
BF	bearing fault	[7] [12] [52] [55]	[51]	[6] [21] [47]
BRBF	broken rotor bar fault	[7] [13] [50]	[16] [51]	[32]
BRF	bowed rotor bar	[14]	[51]	
ChpTooth	chipped tooth		[37]	
CompF	component level fault	[25]		
CrkToothR	crack in tooth root		[37]	
CycSF	cyclic sensor fault			[62]
DCSAF	DC serial arc fault	[57] [58] [59]		
ErrSF	erratic sensor fault			[62]
MisTooth	missing tooth		[37]	
WrnTooth	worn tooth	[37]		
FRSC	faulty ring of squirrel-cage		[16]	
GFOS	gain fault on sensor	[48]		
IDF	irreversive demagnetization fault	[52]	[9] [28]	
ITSF	inter-turn short-circuit fault	[17] [33] [49]	[18] [36]	[8] [10] [20] [39] [41] [42] [45] [46] [66]
IRF	inner raceway fault (BF)	[7] [34] [35] [63] [65]	[28] [37] [51] [56]	[6]
IOSF	intermittent open-switch (OSF)	[4]		
MIF	model inductance fault (MPF)	[27]		
MotF	motor failure	[38] [61]		
MPF	model parameter fault	[31]		
MRF	model resistance fault (MPF)	[1] [27] [49]		
PGSF	phase to ground short-circuit fault	[50]	[36]	[8]
PPSF	phase to phase short-circuit fault	[1] [49] [50]		[8]
OCF	open circuit fault	[2] [29]		[23]
OCPF	open circuit of phase fault	[1]	[36]	
OSF	open switch fault	[40]		
ORF	outer raceway fault (BF)	[7] [34] [35] [60] [63] [65]	[19] [37] [51] [56]	[6] [47]
REF	rotor eccentricity fault	[12] [14]		
RMA	rotor misalignment	[7]	[51]	
RU	rotor unbalance	[34]	[51]	
SCF	short circuit fault	[29]		[23]
SEF	static eccentricity fault			[11] [20]
SM	shaft misalignment	[34]		
SOF	sensor omission fault	[3]		
StckSF	stuck sensor fault			[62]
ApkSF	spike sensor fault			[62]
UA	unidentified anomaly	[15]		

family are often preferred, in particular, if strengthened by ensemble methods. SVM, remains a reference model, the community praises the good performance and low computational cost provided the volume of data is limited; yet another reason for its application. Deep learning techniques have also recently populated the field. The tried-and-true CNN family is the natural choice, whether in their 1-D or 2-D variants. Its application is further facilitated by the availability of public pre-trained models over which transfer learning techniques can be applied. The autoencoder family and deep stacking models are also present. Despite our partial coverage, the diversity of models and methodologies is already apparent.

Table 4 Publications according to AI technique included in the study.

Code	AI technique	RC	EC	BC
ABOD	angle-based outlier detector	[15]		
ADCNN	adaptive deep CNN		[56]	[24]
AE	deep autoencoder	[15]		
AEOD	autoencoder for outlier detection	[15]		
CNN	convolutional neural network	[7] [30] [34] [38] [52] [63] [65]	[28] [56] [19]	
DBN	deep belief network	[25]		[6]
DBNHDN	DBN-based hierarchical diagnosis network		[56]	
DCAE	deep coupling autoencoder		[37]	
DCTLN	deep convolutional transfer learning network		[19]	
DFC	decision forest classifier	[1] [12] [13] [14] [61]		
DJC	decision jungle classifier	[12] [13] [14]		
DTC	decision tree classifier	[1] [3] [12] [17] [38] [50]		
DTR	decision tree regressor	[48]		
EBTC	ensemble bagged trees classifier	[17]		
ESC	ensemble subspace classifier	[17]		
FuzzyL	fuzzy logic	[54]		[21] [46]
FSVM	fuzzy support vector machine			[23]
GBRBM	Gaussian Bernoulli restricted Boltzmann machine		[37]	
GMODBN	Gaussian mixture output dynamic Bayesian network	[60]		
GB	Gradient boosting	[1]		
IN	inception network	[34]		
IDDAN	inferable deep distilled attention network	[63]		
KMC	K-means clustering	[40]		
KNNC	K-nearest neighbor classifier	[1] [3] [17] [50]	[16]	[10] [20]
KNNOD	K-nearest neighbor outlier detector		[15]	
KPCA	kernel PCA			[6]
LLE	locally linear embedding			[6]
LR	logistic regression	[13] [14] [48] [57] [58]	[19] [37]	
MLP	multilayer perceptron	[1] [2] [13] [14] [25] [30] [40] [48] [49] [54] [65]	[16]	[11] [24] [39] [62] [66]
NPE	neighborhood preserving embedding			[6]
TLNN	time-lagged neural network			[42]
OCSVM	one-class SVM			[32]
PDSRC	part dictionary sparse representation classification			[47]
PCA	principal component analysis	[3] [4] [12] [17] [50]	[18]	
PN	prototypical network	[55]		
PSO	particle swarm optimization			[21] [42] [47]
QDA	quadratic discriminant analysis	[52]		
RBFN	radial basis function network			[24]
RBM	restricted Boltzmann machine		[37]	[6]
RNN	recurrent neural network			[41] [45]
ResNN	residual neural network	[34]		
SAE	sparse deep autoencoder		[28] [51]	[6]
SF	sparse filtering		[56]	
SOM	self-organizing map	[40]		[8]
Sperc	simple perceptron	[25]		
SVM	support vector machine	[1] [3] [17] [33] [35] [40] [50] [52] [55] [57] [58] [59] [63] [64]	[51] [56]	[21] [23] [32] [47]
SVR	support vector regression	[30]	[9]	
WCBC	word-code-based classification		[16]	
WDCNN	deep CNN with wide first-layer kernel	[55]		
WPNF	wavelet-prototypical network based on fusion of time and frequency domain	[55]		

To what extent is it feasible to develop DFs implemented with unsupervised ML techniques that rely exclusively on unlabeled data corresponding with the healthy behavior of the system?

We found evidence of systems able to detect anomalies based on machine learning techniques that were trained purely on the healthy behavior of the system. In [15], Geglio et al. apply *reconstruction-based* anomaly detection. A reconstruction-based anomaly detector can be implemented on any signal corresponding to the healthy behavior of the system. They include several variants in their study, a fully convolutional autoencoder (AE CNN), a fully connected autoencoder for outlier detection (AEOD), an angle-based outlier detector (ABOD), and a K-nearest neighbor outlier detector (KNNOD). They also showed that these techniques can integrate the signals of 58 different sensors, that were available in the vehicle's powertrain network. In [37], Ma et al. applied a deep coupling autoencoder (DCAE) to integrate signals from different sensor modalities. The architecture stacked (1) Gaussian Bernoulli restricted Boltzmann machines (GBRBMs); (2) restricted Boltzmann machines (RBMs); (3) a coupling autoencoder (CAE) to integrate vibration and acoustic information; and a (4) multinomial logistic regression classifier taking as input the sparse representation computed by the DCAE architecture. This architecture effectively diagnosed several gearboxes (GB) defects, and BFs outperforming decision pathways based on DAEs operating with the single modalities and both modalities trained independently. By removing the multilayer regressor, reconstruction-based anomaly detection for multimodal signals is possible.

Some solutions are capable of multi-sensor fusion. Chen and Li in [6] apply sparse autoencoders (SAEs) to fuse simultaneously acquired vibration signals from three accelerometers monitoring a motor system. Their proposed method first computes a set of 15 time-based features and 3 frequency-based indicators per sensor. These are fed into a bank of 18 SAEs, each processing the individual features computed from the three vibration sensors. The stacked SAE layers first compute a sparse overcomplete representation, and then a single channel fusion. The 18 fused feature signals are then connected to a deep belief network (DBN) that diagnoses IRFs and ORFs of different magnitudes (slight, moderate, severe). Chen and Li benchmarked SAE-DBN architecture against PCA, KPCA, NPE, LLE, single SAE[3], and AE[4]. SAE-DBN outperformed all other ML models in the benchmark. In addition, it showed better generalization ability and robustness, requiring fewer training samples to achieve high accuracy across various operating points. The SAE stage in the model is an instance of a suitable candidate to be used in unsupervised learning scenarios.

Another example is given by [30], where Kaviya et al. implement condition monitoring (CM) for motors. Taking as inputs the DQ voltages and currents, coolant temperature, stator tooth temperature, temperature of the permanent magnet, motor speed, ambient temperature, and motor torque, they train an SVR to learn to predict a healthy motor's torque. Luo et al. [36] use the three-phase currents and rotor position as inputs to a LSTM network. The network is trained in unsupervised mode to predict the behavior of the healthy system. They show a qualitative confirmation of meaningful residuals that could be used for fault detection of OCPFs, PGSFs, and ITSFs under several operational conditions. There are instances of hybrid architectures that optimize the decisions, for example, in [41], Nyanteh et al. combine a recurrent neural network (RNN) with particle swarm optimization (PSO) to perform fault detection and fault localization. The RNN takes as inputs the three consecutive

[3] Referred to as SAE0 in the paper, is a single SAE, whose input was formed by concatenating the multi-sensor (3×18) features into a single vector.

[4] AE has the same architecture as SAE, but for the sparsity regularization term that is set to zero.

samples of the current phase A of the PMSM (1 input and 2 delays) and computes the current envelopes for each phase; the network being trained by a PSO algorithm having as its objective function an index that measures the ability of the RNN to compute an envelope to the input current along an operational cycle. A fault is detected when the differential current exceeds about three standard deviations from its mean value.

5 Discussion

Many works we have surveyed are instances of *diagnosis as classification*, where a ML model is trained in supervised mode with labeled faults. Often, ML model architectures are hybrids of several paradigms and/or architectures. For instance, the work of Zhang et al. [63] presents the inferable deep distilled attention network (IDDAN). It includes an attention mechanism implemented with a transformer encoder connected to a linear classifier. The authors use *data augmentation*, *knowledge distillation*, and *transfer learning*. A CNN (VGG-16) is trained as the teacher network and IDDAN as the student in the methodology of knowledge distillation. Transfer learning is applied by fine-tuning a multi-layer perception (MLP) fed by the distilled network transformer encoder.

We identify challenging circumstances regarding the setup of experimental platforms to gather data from physical plants that can simulate realistic driving cycles of EVs and excite the system sufficiently across its entire nominal envelope. In many cases, the community relies on public datasets and or synthetic data obtained from systematic simulations. As for simulation frameworks, a notable mention is the work of Meckel et al. [38]. In their study they developed a fault injection generic simulation framework for HEVs implemented in MATLAB/Simulink. Their framework incorporates several fault injection mechanisms allowing faults to manifest transiently, intermittently, or permanently.

Regarding distributed architectures, Jeong et al. [25] engineered an integrated self-diagnosis system (ISS) for autonomous vehicles. Their contribution is a distributed diagnostic solution that trains and optimizes diagnosis models in the cloud using the vehicle's operational data and deploys them at the edge (onboard the vehicle). It consists of three modules: the deep learning module (ODLM), the in-vehicle gateway module, and the edge computing-based self-diagnosis service. The first module performs diagnostics and risk assessment. The second optimizes the data transfer across the networks, collecting data from the vehicle sensors and top-down control signals and routing data to the OBD and actuators. The third informs diagnostic results to the cloud, infrastructure, and other vehicles.

Given the communication infrastructure of modern electrified vehicles, it seems essential to decompose diagnostic tasks into fine-grained computational graphs rather than relying on monolithic solutions. These graphs self-coordinate and coordinate with others, maintaining functionality even under severely degraded conditions. We refer again to the work of Meckel et al. [38] to illustrate two main aspects or requirements for IDS in our domain: (1) active diagnosis , and (2) explicit models of diagnostic procedures. They hypothesized the possibility of transforming a decision tree classifier (DTC) into a diagnosis model in the form of a *diagnostic directed acyclic graph* (DDAG). DDAGs can represent the orchestration of active and distributed processing and feature extraction tasks required in fault diagnosis. In [26], Jo et al. illustrate the use of DDAG for an abstract diagnosis model of the powertrain of an HEV, they showed how as nodes of the graph are executed they gather *evidence* to support a diagnosis decision. The outcome of some nodes (diagnostic tasks) can represent intermediate conclusions. Generally, several factors need to be considered to reach a diagnostic conclusion.

6 Concluding remarks

The landscape of data-driven intelligent diagnosis methods and architectures for EPs is diverse and blooming. In our first installment, of a comprehensive literature review still in the making, we have presented different views and classifications in the hope that they help the community to navigate the arena. The community could benefit from more open data repositories. The digital twin paradigm embraces the integration of multiple techniques, in particular model-based and data-driven approaches. Signal-based methods are already part of many feature-processing pipelines, often combined with deep learning models. Active diagnosis appears to be an essential function of IDSs.

References

1 M Aishwarya and R M Brisilla. Design and Fault Diagnosis of Induction Motor Using ML-Based Algorithms for EV Application. *IEEE Access*, 11:34186–34197, 2023. doi:10.1109/ACCESS.2023.3263588.

2 Mustafa Aktas and Hilmi Aygun. Comparison of DC link current and stator phase current in inverter switching faults detection of PMSM drives in HEVs. *Engineering Science and Technology, an International Journal*, 21(4):664–671, August 2018. doi:10.1016/j.jestch.2018.06.002.

3 Ritik Argawal, Dattatraya Kalel, M Harshit, Arun D Domnic, and R Raja Singh. Sensor Fault Detection using Machine Learning Technique for Automobile Drive Applications. In *2021 National Power Electronics Conference (NPEC)*, pages 1–6, 2021. doi:10.1109/NPEC52100.2021.9672546.

4 Baghli, Delpha, Diallo, Hallouche, Mba, and Wang. Three-Level NPC Inverter Incipient Fault Detection and Classification using Output Current Statistical Analysis. *Energies*, 12(7):1372, April 2019. doi:10.3390/en12071372.

5 Yong Chen, Siyuan Liang, Wanfu Li, Hong Liang, and Chengdong Wang. Faults and Diagnosis Methods of Permanent Magnet Synchronous Motors: A Review. *Applied Sciences*, 9(10):2116, May 2019. doi:10.3390/app9102116.

6 Zhuyun Chen and Weihua Li. Multisensor Feature Fusion for Bearing Fault Diagnosis Using Sparse Autoencoder and Deep Belief Network. *IEEE Transactions on Instrumentation and Measurement*, 66(7):1693–1702, 2017. doi:10.1109/TIM.2017.2669947.

7 Anurag Choudhary, Tauheed Mian, Shahab Fatima, and B K Panigrahi. Deep Transfer Learning Based Fault Diagnosis of Electric Vehicle Motor. In *2022 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, pages 1–6. IEEE, December 2022. doi:10.1109/PEDES56012.2022.10080274.

8 Cao Chuang, Zhang Wei, Wang Zhifu, and Li Zhi. The Diagnosis Method of Stator Winding Faults in PMSMs Based on SOM Neural Networks. *Energy Procedia*, 105:2295–2301, 2017. 8th International Conference on Applied Energy, ICAE2016, 8-11 October 2016, Beijing, China. doi:10.1016/j.egypro.2017.03.663.

9 Renjie Dou, Fangzhen Song, Haining Liu, and Xiuhua Men. Demagnetization Quantification of PMSM Based on Support Vector Regression. In *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pages 619–623, 2018. doi:10.1109/PHM-Chongqing.2018.00111.

10 Bashir Mahdi Ebrahimi and Jawad Faiz. Feature Extraction for Short-Circuit Fault Detection in Permanent-Magnet Synchronous Motors Using Stator-Current Monitoring. *IEEE Transactions on Power Electronics*, 25(10):2673–2682, 2010. doi:10.1109/TPEL.2010.2050496.

11 Bashir Mahdi Ebrahimi, Jawad Faiz, M. Javan-Roshtkhari, and A. Zargham Nejhad. Static Eccentricity Fault Diagnosis in Permanent Magnet Synchronous Motor Using Time Stepping Finite Element Method. *IEEE Transactions on Magnetics*, 44(11):4297–4300, 2008. doi:10.1109/TMAG.2008.2001534.

12 Hicham El Hadraoui, Oussama Laayati, Adila El Maghraoui, Erroumayssae Sabani, Mourad Zegrari, and Ahmed Chebak. Diagnostic and Prognostic Health Management of Electric Vehicle Powertrains : A Data Driven Approach for Induction Motor. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6. IEEE, July 2023. doi:10.1109/ICECCME57830.2023.10253328.

13 Hicham El Hadraoui, Oussama Laayati, Nasr Guennouni, Ahmed Chebak, and Mourad Zegrari. A data-driven Model for Fault Diagnosis of Induction Motor for Electric Powertrain. In *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, pages 336–341. IEEE, June 2022. doi:10.1109/MELECON53508.2022.9843046.

14 Hicham El hadraoui, Mourad Zegrari, Adila El Maghraoui, Oussama Laayati, Erroumayssae Sabani, and Ahmed Chebak. Data-driven Diagnostics for Electric Traction Systems: A Study of Induction Motor. In *IEEE EUROCON 2023 - 20th International Conference on Smart Technologies*, pages 626–631. IEEE, July 2023. doi:10.1109/EUROCON56442.2023.10199047.

15 Anthony Geglio, Eisa Hedayati, Mark Tascillo, Dyche Anderson, Jonathan Barker, and Timothy C Havens. Deep Convolutional Autoencoder for Assessment of Drive-Cycle Anomalies in Connected Vehicle Sensor Data. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 743–749, 2022. doi:10.1109/SSCI51031.2022.10022233.

16 Adam Glowacz. Acoustic based fault diagnosis of three-phase induction motor. *Applied Acoustics*, 137:82–89, 2018. doi:10.1016/j.apacoust.2018.03.010.

17 Konrad Górny, Piotr Kuwałek, and Wojciech Pietrowski. Increasing electric vehicles reliability by non-invasive diagnosis of motor winding faults. *Energies*, 14(9):2510, 2021. doi:10.3390/en14092510.

18 Frank Landry Tanenkeu Guefack, Aleksej Kiselev, and Alexander Kuznietsov. Improved Detection of Inter-turn Short Circuit Faults in PMSM Drives using Principal Component Analysis. In *2018 International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM)*, pages 154–159, 2018. doi:10.1109/SPEEDAM.2018.8445403.

19 Liang Guo, Yaguo Lei, Saibo Xing, Tao Yan, and Naipeng Li. Deep Convolutional Transfer Learning Network: A New Method for Intelligent Fault Diagnosis of Machines With Unlabeled Data. *IEEE Transactions on Industrial Electronics*, 66(9):7316–7325, 2019. doi:10.1109/TIE.2018.2877090.

20 Reemon Z. Haddad and Elias G. Strangas. Detection of static eccentricity and turn-to-turn short circuit faults in permanent magnet synchronous AC machines. In *2015 IEEE 10th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*, pages 277–283, 2015. doi:10.1109/DEMPED.2015.7303702.

21 Hossein Hassani, Jafar Zarei, Mohammad Mehdi Arefi, and Roozbeh Razavi-Far. zSlices-Based General Type-2 Fuzzy Fusion of Support Vector Machines With Application to Bearing Fault Detection. *IEEE Transactions on Industrial Electronics*, 64(9):7210–7217, 2017. doi:10.1109/TIE.2017.2688963.

22 Nikolas Hill, Sophia Amaral, Samantha Morgan-Price, Tom Nokes, Judith Bates, Hinrich Helms, Horst Fehrenbach, Kirsten Biemann, Nabil Abdalla, Julius Jöhrens, Eloise Cotton, Lizzie German, Anisha Harris, Sabine Ziem-Milojevic, Sebastien Haye, Chris Sim, and Ausilio Bauen. Determining the environmental impacts of conventional and alternatively fuelled vehicles through LCA. Technical Report Ricardo/ED11344/Final report/Issue Number 3, European Commission, United Kingdom, July 2020.

23 Jing Hou, Yuan Wang, Tian Gao, and Yan Yang. Fault feature extraction of power electronic circuits based on sparse decomposition. In *2016 International Conference on Condition Monitoring and Diagnosis (CMD)*, pages 505–508, 2016. doi:10.1109/CMD.2016.7757872.

24 Turker Ince, Serkan Kiranyaz, Levent Eren, Murat Askar, and Moncef Gabbouj. Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks. *IEEE Transactions on Industrial Electronics*, 63(11):7067–7075, 2016. doi:10.1109/TIE.2016.2582729.

25 YiNa Jeong, SuRak Son, EunHee Jeong, and ByungKwan Lee. An Integrated Self-Diagnosis System for an Autonomous Vehicle Based on an IoT Gateway and Deep Learning. *Applied Sciences*, 8(7):1164, July 2018. doi:10.3390/app8071164.

26 Seungbum Jo, Markus Lohrey, Damian Ludwig, Simon Meckel, Roman Obermaisser, and Simon Plasger. An architecture for online-diagnosis systems supporting compressed communication. *Microprocessors and Microsystems*, 61:242–256, 2018. doi:10.1016/j.micpro.2018.06.008.

27 Daniel Jung and Qadeer Ahmed. Active Fault Management in Autonomous Systems Using Sensitivity Analysis. *IFAC-PapersOnLine*, 51(24):1099–1104, 2018. doi:10.1016/j.ifacol.2018.09.730.

28 I-Hsi Kao, Wei-Jen Wang, Yi-Horng Lai, and Jau-Woei Perng. Analysis of Permanent Magnet Synchronous Motor Fault Diagnosis Based on Learning. *IEEE Transactions on Instrumentation and Measurement*, 68(2):310–324, 2019. doi:10.1109/TIM.2018.2847800.

29 Halid Kaplan, Kambiz Tehrani, and Mo Jamshidi. A Fault Diagnosis Design Based on Deep Learning Approach for Electric Vehicle Applications. *Energies*, 14(20):6599, October 2021. doi:10.3390/en14206599.

30 M R Kaviya, S Hemamalini, G H Grace, and R Srimathi. Torque prediction for condition monitoring of permanent magnet synchronous motor using machine learning algorithms. In *International Conference on Green Energy, Computing and Intelligent Technology (GEn-CITy 2023)*, volume 2023, pages 85–90. Institution of Engineering and Technology, 2023. doi:10.1049/icp.2023.1765.

31 Nassim Khaled. Streamlined Process for Cloud Based Diagnostics Using Amazon Web Services. In *SAE WCX Digital Summit*. SAE International, April 2021. doi:10.4271/2021-01-0159.

32 Jaroslaw Kurek and Stanislaw Osowski. Support vector machine for fault diagnosis of the broken rotor bars of squirrel-cage induction motor. *Neural Computing and Applications*, 19:557–564, 2010. doi:10.1007/s00521-009-0316-5.

33 Siyuan Liang, Yong Chen, Hong Liang, and Xu Li. Sparse Representation and SVM Diagnosis Method Inter-Turn Short-Circuit Fault in PMSM. *Applied Sciences*, 9(2):224, January 2019. doi:10.3390/app9020224.

34 Shih-Lin Lin. Application Combining VMD and ResNet101 in Intelligent Diagnosis of Motor Faults. *Sensors (Basel, Switzerland)*, 21(18):6065, 2021. doi:10.3390/S21186065.

35 Shih-Lin Lin. Application of Machine Learning to a Medium Gaussian Support Vector Machine in the Diagnosis of Motor Bearing Faults. *Electronics*, 10(18):2266, September 2021. doi:10.3390/electronics10182266.

36 Yefeng Luo, Jianqi Qiu, and Cenwei Shi. Fault Detection of Permanent Magnet Synchronous Motor Based on Deep Learning Method. In *2018 21st International Conference on Electrical Machines and Systems (ICEMS)*, pages 699–703, 2018. doi:10.23919/ICEMS.2018.8549129.

37 Meng Ma, Chuang Sun, and Xuefeng Chen. Deep Coupling Autoencoder for Fault Diagnosis With Multimodal Sensory Data. *IEEE Transactions on Industrial Informatics*, 14(3):1137–1145, 2018. doi:10.1109/TII.2018.2793246.

38 Simon Meckel, Tim Schuessler, Pravin Kumar Jaisawal, Jie-Uei Yang, and Roman Obermaisser. Generation of a diagnosis model for hybrid-electric vehicles using machine learning. *Microprocessors and Microsystems*, 75:103071, June 2020. doi:10.1016/j.micpro.2020.103071.

39 S.S. Moosavi, A. Djerdir, Y. Ait-Amirat, and D.A. Khaburi. ANN based fault diagnosis of permanent magnet synchronous motor under stator winding shorted turn. *Electric Power Systems Research*, 125:67–82, 2015. doi:10.1016/j.epsr.2015.03.024.

40 S.S. Moosavi, A Kazemi, and H Akbari. A comparison of various open-circuit fault detection methods in the IGBT-based DC/AC inverter used in electric vehicle. *Engineering Failure Analysis*, 96:223–235, February 2019. doi:10.1016/j.engfailanal.2018.09.020.

41 Yaw Nyanteh, Chris Edrington, Sanjeev Srivastava, and David Cartes. Application of artificial intelligence to real-time fault detection in permanent-magnet synchronous machines. *IEEE Transactions on Industry Applications*, 49(3):1205–1214, 2013. doi:10.1109/TIA.2013.2253081.

42 Yaw D. Nyanteh, Sanjeev K. Srivastava, Chris S. Edrington, and David A. Cartes. Application of artificial intelligence to stator winding fault diagnosis in permanent magnet synchronous machines. *Electric Power Systems Research*, 103:201–213, 2013. doi:10.1016/j.epsr.2013.05.018.

43 Society of Automotive Engineers. SAE J3016 Standard. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. https://www.sae.org/standards/content/j3016_202104/, 2021.

44 Society of Automotive Engineers. SAE J3063 Active Safety Systems Terms and Definitions. https://www.sae.org/standards/content/j3063_202305/, 2023.

45 J. Quiroga, D.A. Cartes, C.S. Edrington, and Li Liu. Neural network based fault detection of PMSM stator winding short under load fluctuation. In *2008 13th International Power Electronics and Motion Control Conference*, pages 793–798, 2008. doi:10.1109/EPEPEMC.2008.4635364.

46 J. Quiroga, Li Liu, and D. A. Cartes. Fuzzy logic based fault detection of PMSM stator winding short under load fluctuation using negative sequence analysis. In *2008 American Control Conference*, pages 4262–4267, 2008. doi:10.1109/ACC.2008.4587163.

47 Likun Ren, Weimin Lv, ShiWei Jiang, and Yang Xiao. Fault Diagnosis Using a Joint Model Based on Sparse Representation and SVM. *IEEE Transactions on Instrumentation and Measurement*, 65(10):2313–2320, 2016. doi:10.1109/TIM.2016.2575318.

48 Tunan Shen, Yuping Chen, Christian Thulfaut, and Hans-Christian Reuss. A Data Based Diagnostic Method for Current Sensor Fault in Permanent Magnet Synchronous Motors (PMSM). In *IECON Proceedings (Industrial Electronics Conference)*, volume 2019-Octob, pages 5979–5985, 2019. doi:10.1109/IECON.2019.8927667.

49 Tunan Shen, Ahmet Kilic, Christian Thulfaut, and Hans-Christian Reuss. An Intelligent Diagnostic Method for Permanent Magnet Synchronous Motors (PMSM) in the Electric Drive of Autonomous Vehicles. In *2019 21st European Conference on Power Electronics and Applications (EPE '19 ECCE Europe)*, pages P.1–P.10. IEEE, September 2019. doi:10.23919/EPE.2019.8915161.

50 Andre A. Silva, Shalabh Gupta, Ali M. Bazzi, and Arthur Ulatowski. Wavelet-based information filtering for fault diagnosis of electric drive systems in electric ships. *ISA Transactions*, 78:105–115, 2018. doi:10.1016/j.isatra.2017.08.013.

51 Chuang Sun, Meng Ma, Zhibin Zhao, and Xuefeng Chen. Sparse Deep Stacking Network for Fault Diagnosis of Motor. *IEEE Transactions on Industrial Informatics*, 14(7):3261–3270, 2018. doi:10.1109/TII.2018.2819674.

52 Zia Ullah, Bilal Ahmad Lodhi, and Jin Hur. Detection and Identification of Demagnetization and Bearing Faults in PMSM Using Transfer Learning-Based VGG. *Energies (Basel)*, 13(15):3834, 2020.

53 Adil Usman, Bhakti M Joshi, and Bharat S Rajpurohit. Review of fault modeling methods for permanent magnet synchronous motors and their comparison. In *2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*, pages 141–146, 2017. doi:10.1109/DEMPED.2017.8062347.

54 Suchitra Venkatesan, Krishnan Manickavasagam, Nikita Tengenkai, and Nagendran Vijayalakshmi. Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin. *IET Electric Power Applications*, 13(9):1328–1335, 2019. doi:10.1049/iet-epa.2018.5732.

55 Yu Wang, Lei Chen, Yang Liu, and Lipeng Gao. Wavelet-Prototypical Network Based on Fusion of Time and Frequency Domain for Fault Diagnosis. *Sensors*, 21(4):1483, February 2021. doi:10.3390/s21041483.

56 Long Wen, Xinyu Li, Liang Gao, and Yuyan Zhang. A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method. *IEEE Transactions on Industrial Electronics*, 65(7):5990–5998, 2018. doi:10.1109/TIE.2017.2774777.

57 Kun Xia, Haotian Guo, Sheng He, Wei Yu, Jingjun Xu, and Hui Dong. Binary classification model based on machine learning algorithm for the DC serial arc detection in electric vehicle battery system. *IET power electronics*, 12(1):112–119, 2019.

58 Kun Xia, Bangzheng Liu, Xiale Fu, Haotian Guo, Sheng He, Wei Yu, Jingjun Xu, and Hui Dong. Wavelet entropy analysis and machine learning classification model of DC serial arc fault in electric vehicle power system. *IET Power Electronics*, 12(15):3998–4004, December 2019. doi:10.1049/iet-pel.2019.0375.

59 Kun Xia, Zihan Zhang, Bangzheng Liu, Haotian Guo, Wei Yu, Jingjun Xu, and Hui Dong. Data-enhanced machine recognition model of DC serial arc in electric vehicle power system. *IET Power Electronics*, 13(19):4677–4684, December 2020. doi:10.1049/iet-pel.2020.0923.

60 Hongtao Xue, Jiawen Zhou, Zhenyu Chen, and Zhongxing Li. Real-Time Diagnosis of an In-Wheel Motor of an Electric Vehicle Based on Dynamic Bayesian Networks. *IEEE Access*, 7:114685–114699, 2019. doi:10.1109/ACCESS.2019.2935770.

61 Raziq Yaqub, Hassan Ali, and Mohd Helmy Bin Abd Wahab. Electrical Motor Fault Detection System using AI's Random Forest Classifier Technique. In *2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, pages 1–5. IEEE, April 2023. doi:10.1109/IC_ASET58101.2023.10150924.

62 Hongkun Zhang and Wenjun Li. A New Method of Sensor Fault Diagnosis Based on a Wavelet Packet Neural Network for Hybrid Electric Vehicles. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1143–1147, 2016. doi:10.1109/CISP-BMEI.2016.7852886.

63 Xiaotian Zhang, Yihua Hu, Aijun Yin, Jiamei Deng, Hui Xu, and Jikai Si. Inferable Deep Distilled Attention Network for Diagnosing Multiple Motor Bearing Faults. *IEEE Transactions on Transportation Electrification*, 9(2):2207–2216, June 2023. doi:10.1109/TTE.2022.3211203.

64 Chunheng Zhao, Yi Li, Matthew Wessner, Chinmay Rathod, and Pierluigi Pisu. Support-Vector Machine Approach for Robust Fault Diagnosis of Electric Vehicle Permanent Magnet Synchronous Motor. In *Annual Conference of the PHM Society*, volume 12(1), page 10, November 2020. doi:10.36001/phmconf.2020.v12i1.1291.

65 Ping Zou, Baocun Hou, Lei Jiang, and Zhenji Zhang. Bearing Fault Diagnosis Method Based on EEMD and LSTM. *International Journal of Computers, Communications and Control*, 15(1), February 2020. doi:10.15837/IJCCC.2020.1.3780.

66 Ferhat Çira, Müslüm Arkan, and Bilal Gümüş. A new approach to detect stator fault in permanent magnet synchronous motors. In *2015 IEEE 10th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*, pages 316–321, 2015. doi:10.1109/DEMPED.2015.7303708.

Data-Driven RUL Prediction Using Performance Metrics

Abel Diaz-Gonzalez
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Austin Coursey
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Marcos Quinones-Grueiro
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Chetan S. Kulkarni
NASA Ames Research Center (KBR, Inc), Moffett Field, CA, USA

Gautam Biswas
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

Abstract

Prognostics is the scientific study of component and system degradation with use, and the prediction of when failure may occur. In this work, we propose a new data-driven method for predicting a system's remaining useful life (RUL) without needing an accurate system model or expert knowledge. Instead, we use system operational data to estimate how the system's performance metrics change with time. Although this is a purely data-driven approach, the method's design is inspired by model-based techniques. First, we frame a novel Multitask Machine Learning architecture to simultaneously learn the general pattern of performance degradation and the individual trajectories from run-to-failure performance trajectory data. We apply this method to the set of performance metrics that determine the system's end-of-life (EOL), building a performance trajectory library of the system operation under different operational conditions. We leverage the performance metric library as prior belief and develop a Bayesian deep learning approach to update the performance measures over time and predict the system EOL. We evaluate our method on two datasets of the N-CMAPSS benchmark, achieving satisfactory results in terms of overall performance and uncertainty estimation accuracy. Overall, our approach illustrates a generalized deep learning architecture that can more effectively predict the system RUL for a collection of identical systems.

2012 ACM Subject Classification Applied computing → Aerospace; Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning approaches

Keywords and phrases remaining useful life, data-driven methods, machine learning, performance metric, multitask machine learning, Monte Carlo

Digital Object Identifier 10.4230/OASIcs.DX.2024.21

Category Short Paper

1 Introduction

Industrial systems and their components experience degradation over time with a risk of experiencing failure. To perform condition-based maintenance (CBM), operators must use prognostics approaches to estimate the remaining useful life (RUL) of these systems and their components. Accurate estimation of the RUL allows operators to make informed decisions about system health and maintenance decisions thus increasing the resilience of real-world system operations [15]. Techniques for estimating RUL can be divided into three major categories: model-based, data-driven, and hybrid [8].



© Abel Diaz-Gonzalez, Austin Coursey, Marcos Quinones-Grueiro, Chetan S. Kulkarni, and Gautam Biswas;
licensed under Creative Commons License CC-BY 4.0



Model-based techniques leverage system physics to develop mathematical models for prognostics, achieving notable success. For instance, [5] presents a method estimating state and damage progression parameters, then predicting the end of life (EOL) by propagating these estimates. Applied to a centrifugal pump model, they found the unscented Kalman filter optimal for state-parameter estimation, though other filters might suit different applications [6]. Recent advancements include extending classical physics-based prognostics with mechanistic approaches for accurate RUL estimation in lithium-ion batteries, albeit with higher computational costs [7]. These techniques are powerful when data is limited but require expert knowledge, which may be impractical for complex systems [4].

To overcome modeling challenges, data-driven techniques have been developed, learning mappings from measured variables to RUL without detailed expert input. Deep learning methods, particularly recurrent neural networks (RNNs), are popular in aircraft engine prognostics [19]. For instance, [9] employs a bidirectional long short-term memory network on the C-MAPSS dataset. However, RNNs struggle with long sequences due to the vanishing gradient problem [14], and even advanced RNNs lack parallelization [18]. Transformers, like the temporal flow transformer for bearing RUL estimation [3] and methods predicting proton exchange membrane fuel cell degradation [11], have shown superior performance. Other popular approaches include convolutional neural networks fused with LSTMs [12], Gaussian processes [2], and multi-layer perceptrons [17].

Data-driven methods, despite their popularity, have limitations such as lack of explainability [8] and the need for large, representative datasets [4]. Addressing these, [21] and [13] incorporate uncertainty quantification in RUL estimation. Hybrid approaches combining data-driven and model-based methods have been proposed, such as fusing data-driven features with degradation models [20] or using physics-derived features in deep learning models [4].

This paper introduces a novel approach inspired by model-based prognostics [10] but adopts an entirely data-driven approach. We reformulate model-based components using data-driven models, offering explainability without requiring expert knowledge. Our method frames performance degradation modeling as a multitask machine-learning problem, creating a data-driven performance trajectory library. This library is used to predict EOL by solving the uncertainty propagation problem for each performance metric and updating prior beliefs with new data. Evaluated on two N-CMAPSS sub-datasets for aircraft engine prognostics [1], our contributions are:

- An end-to-end data-driven prognostics approach estimating RUL and associated uncertainty based on performance metrics, independent of system physics.
- Demonstration of the framework's potential on N-CMAPSS datasets, achieving satisfactory predictive results and outlining future research directions.

The rest of the paper is structured as follows. Section 2 provides an explanation and the theoretical foundations of the proposed method. Subsection 2.2 focuses on the training process, while the second step, responsible for predicting the system's RUL, is addressed in Subsection 2.3. Section 3 presents an analysis of the experimental results and Section 4 concludes with a summary of the main findings and outlines future work.

2 Proposed Approach

Our proposed approach consists of two main steps. As a first step, we learn a library of run-to-failure trajectory distributions for each performance metric using training data. We create a new *Multitask Machine Learning model* for each performance metric. Each training trajectory corresponding to a performance metric constitutes a learning task, and all trajectories share the general performance pattern as common information. Learning this library constitutes the offline training phase of our method as shown in Figure 1a.

Our second step uses the learned trajectory distribution libraries as prior beliefs for predicting the end of life (EOL) corresponding to each performance metric. This is illustrated in Figure 1b. The procedure to generate the predictions associated with each performance metric is shown in Figure 1c and can be summarized as follows:

1. **First Prediction:** At time step $n = 0$, before receiving observations, we make an initial prediction for each performance metric by sampling the performance trajectories in our library uniformly n_{paths} times, where n_{paths} is a hyperparameter of the method indicating the number of predictions we make at each time step. Then, we compute the EOL and uncertainty boundaries of the system RUL from these initial predictions.
2. For each time step $n \geq 1$ repeat:
   a. **Performance Prediction.**
      For each Performance metric $i = 1, 2, \ldots, \kappa$, repeat:
      i. **Correction Step (see Section 2.3.4):** Select the best predictions from the previous time step in the light of the new observation.
      ii. **Build Trajectory Belief (Section 2.3.4):** Select a set of beliefs, or a guess, around the best performance predictions obtained in the previous step.
      iii. **Uncertainty Propagation (Section 2.3.3):** For each belief, predict the performance by finding the best linear combination of the learned trajectory library that maximizes the posterior probability under the given prior belief. The set of optimal linear combinations for each belief gives the set of n_{paths} predicted performance trajectory at time n.
   b. **Compute System RUL(Section 2.3.5):** After computing the n_{paths} performance predictions for each performance metric, we compute the EOL and uncertainty bounds for the system at the current time.

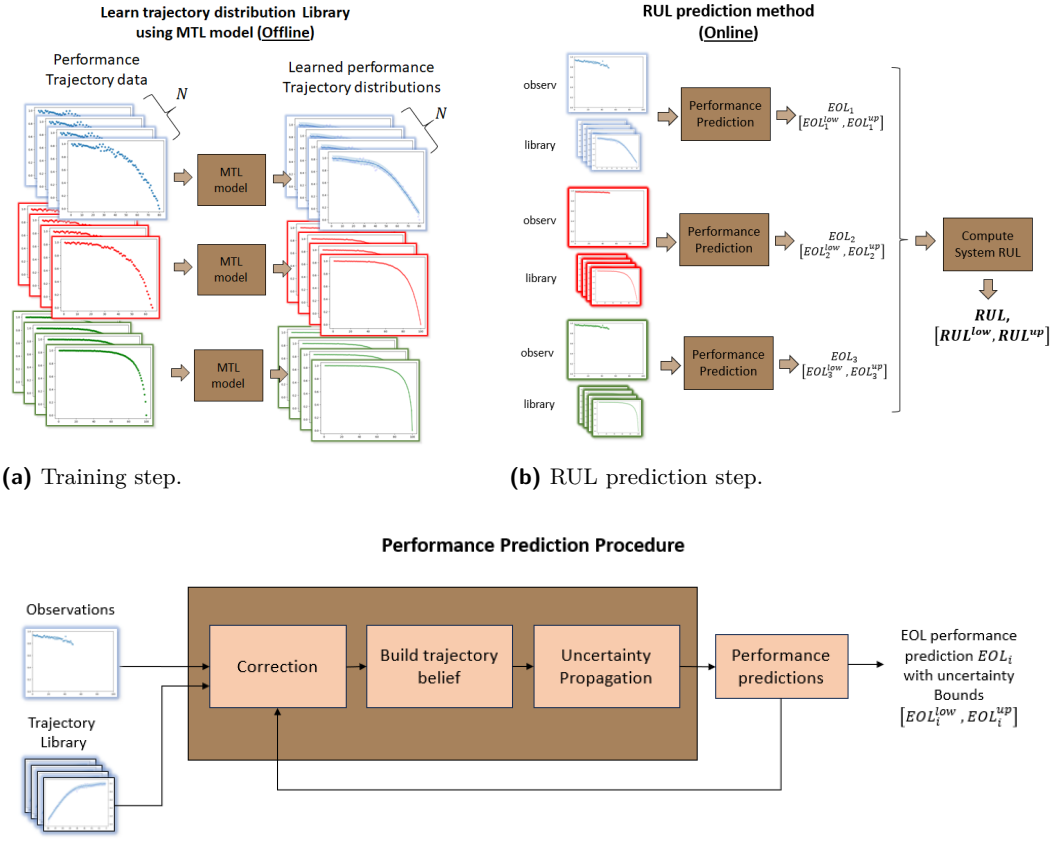
2.1 Training data

Denote by κ the number of performance metrics that determine the EOL of the system. For simplicity, we assume that we record the value of every performance metric throughout its life. Then, if we have the run-to-failure trajectories of N units, our training data consists of the set of N (multivariate) time series of performance metric trajectories $\{u_i\}_{i=1}^{N}$, where $u_i = \{(t_j, p_j)\}_{j=1}^{L_i}$, $p_j = (p_j^{(1)}, p_j^{(2)}, \ldots, p_j^{(\kappa)})$ is the vector of performance metric values collected at time t_j, and L_i denotes the number of time steps until the EOL of the i^{th} training unit. Notice that we are omitting the unit indexation to simplify the notation. Our approach also needs a maximum life value $t_{\max}$ that determines the maximum life we are considering. Then $t_1 < t_2 < \cdots < t_{L_i} \in [0, t_{\max}]$ for every training unit and the EOL of the i^{th} unit is $\text{EOL}_i = t_{L_i}$ for $i = 1, 2, \ldots, N$.

We also assume that the thresholds determining the EOL for each performance metric are constant values independent of time. Therefore, by scaling, we can assume without loss of generality that the performance metrics are decreasing functions from $[0, t_{L_i}] \subset [0, t_{\max}]$ to $[0, 1]$, where a value of 1 indicates a completely healthy performance indicator and a value of 0 means that the performance metric has reached its EOL.

2.2 Learning the Performance Trajectory Library

During the training process, we learn all the run-to-failure trajectories of the same performance metric in parallel (as individual Gaussian stochastic processes) by using a multitask machine learning architecture that learns the performance metric pattern in a block shared along the



(a) Training step.

(b) RUL prediction step.

(c) Performance prediction procedure.

Figure 1 Overview of the proposed method for the case of three performance metrics $\kappa = 3$.

individual trajectory models. Specifically, for each performance metric with N run-to-failure trajectories, we learn a collection of time series $h_i(t) = (\mu_i(t), \sigma_i(t))$ for $i = 1, 2, \ldots, N$, where $\mu_i(t)$ and $\sigma_i(t)$ represent the mean and standard deviation of the i^{th} run-to-failure trajectory at time t. We call this collection of time series h a trajectory library. We model the individual trajectories by using linear transformations of inputs $g^{\text{in}} = (g_1^{\text{in}}, g_2^{\text{in}}, \ldots, g_N^{\text{in}})$ and outputs $g^{\text{out}} = (g_1^{\text{out}}, g_2^{\text{out}}, \ldots, g_N^{\text{out}})$ of the common block, from now on referred to as latent function f in charge of learning the performance pattern. By composing both, we build an individual model for the performance trajectory distribution of each training unit: $h_i(x) = (g_i^{\text{out}} \circ f \circ g_i^{\text{in}})(x), \quad i = 1, 2, \ldots, N$. The functions g_i^{out}, g_i^{in}, and f are learnable. Note that the latent function f and its learnable parameters are common to all the trajectory models h_i. Thus, by learning from all the trajectories, we ensure that the individual trajectory information passes through the latent function f. This enforces f to learn only the common information across trajectories while simultaneously transferring information from long to short trajectories.

We modeled our latent function f as a multilayer perceptron (MLP) neural network with residual connections. The functions $g_i^{\text{in}} : \mathbb{R} \to \mathbb{R}$ and $g_i^{\text{out}} : \mathbb{R}^2 \to \mathbb{R} \times \mathbb{R}_+$ are linear layers $g_i^{\text{in}}(x) = a_i x + b_i$ and $g_i^{\text{out}}(x_1, x_2) = (\mu(x_i), \sigma(x_2)) = (c_i^{\text{mean}} x_1 + d_i^{\text{mean}}, c_i^{\text{std}} x_2 + d_i^{\text{std}})$, with learnable parameters $a_i, b_i, c_i^{\text{mean}}, c_i^{\text{std}}, d_i^{\text{mean}}, d_i^{\text{std}}$ for $i = 1, 2, \ldots, N$. Since the standard deviation is a positive value, we also need to apply a positive function to the second output

of g_i^{out}. In our implementation, we chose the Softplus function ($\text{Softplus}(x) = \log(1 + e^x)$). To introduce monotonicity in the performance metric, we restrict the sign of the slope coefficients of the linear transformation layers. All slopes are positive except for the slope of the output layer associated with the mean, whose sign depends on the monotonicity of our performance metric. Assuming our performance is monotonically decreasing, we restrict the slope coefficients as $a_i, c_i^{\text{std}} > 0$ and $c_i^{\text{mean}} < 0$ for $i = 1, 2, \ldots, N$. We also introduce monotonicity in our latent function f by adding a term in the loss function of our model that strongly penalizes a negative derivative of f. The monotonicity of the entire model (mean and standard deviation outputs) comes then from the monotonicity of the composition of monotonic functions. Notice that we are also imposing the standard deviation to be non-decreasing, assuming that the degradation of the model does not reduce the variability of our data.

The loss function to learn the trajectory performance library consists of three terms. The main term is in charge of learning the trajectory distributions of all training units, and the second and third terms enforce the monotonicity and smoothness of the latent function:

$$\sum_{i=1}^{N} \text{NLL}(h_i(t_1, t_2, \ldots, t_{L_i}) | p_1, p_2, \ldots, p_{L_i}) + \frac{\gamma_1}{M} \sum_{i=1}^{M} \max(f'(s_i), 0)^2 + \frac{\gamma_2}{M} \sum_{i=1}^{M} (f'''(s_i))^2,$$

where NLL denotes the negative log-likelihood of the normal distribution and $s_1 < s_2 < \cdots < s_M$ is a uniform partition of the minimal interval containing all the outputs of the training points through the input linear transformations g_i^{in}.

We also assume that the performance metric values are independent over time. Although performance values that are close in time should be correlated, we make this assumption for simplicity and to reduce the computational cost. An alternative is to model the covariance between different points in time with a kernel function like the radial basis function kernel used in Gaussian Processes. We plan to address this in future work.

2.3 RUL Prediction method

Our RUL prediction method is inspired by the model-based RUL prediction technique, where we first estimate the current state of the system, and then, using a transition model, we perform a Monte Carlo Simulation to predict the EOL of the system [10]. In our case, instead of a transition model for the system, we use training data of the performance metrics of the system. At a high level, our model estimates the current state for every performance metric by using the history of observations until the current time. It then predicts the EOL of each performance metric by combining this information with the library of performance distributions to predict their future evolution. This is done by performing an uncertainty propagation procedure to predict the set of possible ways the performance observation could evolve in time based on the observations and our library of trajectories learned from data. In this way, our method computes the possible EOL of each performance metric and the RUL of the system with uncertainty bounds.

2.3.1 Linear Space for Trajectory Searching

Our method is based on combining the present information that comes from the set of observations of the performance metric obtained until the current time step n with the future information that comes from the library of trajectory distributions learned from data $\{h_1, h_2, \ldots, h_N\}$. We combine both by finding the linear combination of our trajectory library

distributions that "best" fits our observations. Then, we consider the space of all possible trajectory distributions as the linear space generated by the set of distributions in our library

$$\mathcal{L}(h) = \left\{ \sum_{i=1}^{N} \alpha_i h_i : \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N) \in \mathbb{R}^N \right\}.$$

Note that this space contains many distributions that have no practical meaning. For example, if we restrict ourselves to the mean of the distributions, the constant function $g \equiv 0$ is always part of the space, independent of the library we have. Also, the linear combinations where all scalars are negative are defined in $\mathcal{L}(h)$, but this inverts the monotonicity of the resulting performance metric. We could use a more restrictive space that ensures more reasonable performance distributions like the space of convex combinations, also known as convex hull, of the (function) vectors $h = (h_1, h_2, \ldots, h_N)$, but this space is too restrictive for our purposes. We decided to keep our searchable trajectories as the linear space $\mathcal{L}(h)$ and avoid unrealistic trajectories through an intelligent search.

Notice that this is a linear space of Gaussian distributions. Since we trained our trajectory library to find the best Gaussian distribution fitting our training data, we know that $h_i = (\mu_i, \sigma_i)$ distributes $\mathcal{N}(\mu_i(t), \sigma_i^2(t)|t)$ and, under the common assumption of training trajectories independence, we obtain that the linear linear combination $\tilde{h} = \sum_{i=1}^{N} \alpha_i h_i \in \mathcal{L}(h)$ is also Gaussian with distribution

$$\mathcal{N}\left(\sum_{i=1}^{N} \alpha_i \mu_i(t), \sum_{i=1}^{N} \alpha_i^2 \sigma_i^2(t) \middle| \, t \right).$$

2.3.2 Finding the best Linear Combination

We formulate this problem by finding the best vector of scalars α that maximizes the α-parameterized likelihood of the observations. As we mentioned when we learned the trajectory library, we also assume here independence over time for the same reasons.

Once we know the joint distribution of the linear combination of our library distributions (as a function of the scalars α), we find the scalar values α of the linear combination that best fit the observations until the current time by maximizing the likelihood of the observation, or, equivalently, minimizing its negative log-likelihood.

In our problem, the parameters we are optimizing are the scalars of the linear combination α. If we assume a Gaussian probability prior with mean $\bar{\alpha}$ and covariance matrix $\frac{1}{\gamma} I$, where γ quantifies how much we believe that the right linear combination α is close to $\bar{\alpha}$,

$$\mathrm{NLL}_{\bar{\alpha},\gamma}(\alpha|\mathbf{p}_n) = \mathrm{NLL}(\alpha|\mathbf{p}_n) + \gamma(\alpha - \bar{\alpha})^T(\alpha - \bar{\alpha}) = \mathrm{NLL}(\alpha|\mathbf{p}_n) + \gamma\|\alpha - \bar{\alpha}\|^2, \tag{1}$$

where $\|\cdot\|$ denotes the euclidean norm in $\mathbb{R}^N$. Notice also that we have omitted the quantities independent of α as they do not affect the optimization.

2.3.3 Uncertainty Propagation

In model-based approaches, Monte Carlo simulations help to compute stochastically the possible ways in which the system may evolve under the influence of inputs and environmental conditions. Since we do not have a transition model, we cannot compute the next state given new inputs. Instead, we compute the most probable ways in which each performance metric may evolve, given the history of observations and a belief about its future evolution, by sampling the trajectories available in the library. Our proposed Uncertainty Propagation method is performed at each time step by solving the optimization problem with an objective function (1) for $\mathcal{N}(\bar{\alpha}, \frac{1}{\gamma} I)$ in a set of beliefs $\tilde{A}_n$ that change dynamically over time.

This belief set $\tilde{A}_n$ is composed of two belief sets, a present belief set A_n^p in charge of tracking the best prediction at the present time, and a future belief set A_n^f, that keeps the library of trajectories as part of our belief space because sampling from them may help us propagate our predictions into the future. Therefore, our projections can accommodate and continue to be guided by future observations as they become available.

We first notice (see (1)) that the contribution of the observations until the present time, which we will call past information, and the present and future information generated by our belief sets should include the effect of the degradation and the environmental conditions at each time step n. At the beginning of the life of the system, when the degradation effects are small, the observations follow a mostly nominal behavior. Therefore, the prediction of the end of life derived from the performance metrics has large uncertainty bounds. Moreover, since we have a few observations of system performance and the time to system end-of-life is in the distant future, we have to rely more on the trajectories available in our library to predict the future performance evolution (therefore, γ is larger). As the system approaches the end of life, we can give much larger weight to the set of observations on system behavior that we have accumulated because they contain much more information about degradation effects and the condition under which the system has operated. These are likely much better predictors of the system end of life than our prior beliefs of the system performance trajectories (implying γ should be smaller). Assuming that the performance metric is decreasing and scaled in the interval $[0, 1]$, we can use the value of the current performance observation p_n to weigh the past-future importance and balance these contributions in computing the EOL and RUL predictions. To reflect this, we modify (1) as follows

$$\text{objective}_{\mathbf{p}_n}(\alpha|\bar{\alpha}, \gamma) = (1 - p_n)\text{NLL}(\alpha|\mathbf{p}_n) + p_n\gamma\|\alpha - \tilde{\alpha}\|^2.$$

Note that, at the beginning of life, the observations reflect healthy performance values, so they are close to 1 and our optimization function mostly relies on our (present and future) beliefs. As the system degrades, the current performance observations, p_n, is used to give more weight to observations when performing the optimizations to compute the EOL.

In addition, we modify $\text{NLL}(\alpha|\mathbf{p}_n)$ to give more importance to fit recent performance observations than observations in time by doing a (linear) weighted sum

$$\text{NLL}(\alpha|\mathbf{p}_n) = \sum_{j=1}^{n} w_{j,n} \left[\frac{[\mathbf{p}_n - U\alpha]_j^2}{[V\alpha^2]_j} + \ln([V\alpha^2]_j) \right]$$

where $w_{i,n} = \text{Softmax}(w_{\max}(1 - p_i))$ for $i = 1, 2, \ldots, n$, $w_{\max}$ is a hyperparameter of the method indicating how much we care about future observation over past observations or the rate at which we *forget* past observations. At every time step n, given a set of beliefs $\tilde{A}_n = A_n^p \cup A_n^f$ and history of observations until current time $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, we obtain the set of predictions of our uncertainty propagation method as

$$\tilde{A}_n^* = \left\{ \arg\min_{\alpha \in \mathbb{R}^N} \{\text{objective}_{\mathbf{p}_n}(\alpha|\bar{\alpha}, \gamma)\} : \mathcal{N}\left(\bar{\alpha}, \frac{1}{\gamma} I\right) \in \tilde{A}_n \right\}.$$

If we obtain predictions that violate the monotonicity of the performance metric in the time interval $[t_n, t_{\max}]$, these predictions are pruned and we repeat the simulation for the number of pruned trajectories. This may happen because of the admission of negative values of the linear combination, so this prune, together with our belief (as we will explain later), enforces our search to be close to the convex hull of h.

2.3.4 Dynamic Beliefs

We already have discussed how to make performance predictions given observations and a belief set $\tilde{A}$, but how do we modify these sets over time? At each time step, the only information we need is the predictions we made in the previous step A_{n-1}^{pred}. Our method starts before we receive any observations ($n = 0$). At this initial state, we are already able to make predictions solely based on our trajectory library, as our Uncertainty Propagation method does when we have observations for $n \geq 1$. We make our first n_{paths} predictions by doing n_{paths} uniform sampling of the performance trajectories h_i, for $i = 1, 2, \ldots, N$. Then, at time step n, and given our previous predictions A_{n-1}^{pred}, we build our current predictions A_n^{pred} inductively as follows:

Correction Step. Once we receive the predictions at the previous time step A_{n-1}^{pred} we correct them in light of the new observation. This will give us the centers around which we will build our belief sets:

$$\alpha^{\text{present}} = \underset{\alpha \in A_{n-1}^{\text{pred}}}{\text{argmin}} \{\text{NLL}(\alpha|\mathbf{p}_n)\}, \quad \alpha_i^{\text{future}} = \underset{\lambda \in \mathbb{R}}{\text{argmin}} \{\text{NLL}(\lambda e_i|\mathbf{p}_n)\} e_i, \quad i = 1, 2, \ldots, N.$$

Notice that the center of our present belief is the prediction in A_{n-1}^{pred} that has the smallest NLL of the observations(including the new observation p_n), while the N centers of the future beliefs are the best scaling of the training trajectories fitting the observations. In order to assign a probability for each center to be selected, we also collect the minimum values of the future centers $q_i = \min_{\lambda \in \mathbb{R}} \{\text{NLL}(\lambda e_i|\mathbf{p}_n)\}$, and convert them into "probabilities" by applying the Softmax function to obtain the vector of probabilities $p^{\text{future}} = \text{Softmax}(q)$, where $q = (q_1, q_2, \ldots, q_N)^T$.

Build the Belief sets. Each belief in our belief sets is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$. The means μ for both belief sets, A_n^p and A_n^f, are constructed by a Gaussian sampling around their centers. The sampling here simulates our selection of different beliefs around a center and has no probabilistic implications in our model other than an automatic and "reasonable" way to select different beliefs. On the other hand, the variance σ^2 only depends on the centers. All the beliefs sampled around the same center share the same variance, σ^2. We refer to the inverse of the variance of a center as the trust in the center, denoted by γ. Since we have one present center and N future centers, we denote their respective trusts by γ^p for the present center and γ_i^f for the i-th future center.

Since our beliefs are on trajectories and we only have access to the scalars that generate these trajectories, adding a Gaussian noise directly in the scalars creates a bias towards the origin of coordinates. But there is no reason to have a stronger belief about the constant zero trajectory, so we make a change of base from h to the mean-centered base $\{h_i - \bar{h}\}_{i=1}^N$, where $\bar{h} = \frac{1}{n} \sum_{i=1}^N h_i$, before applying the Gaussian noise on the scalars. This way we move the origin of coordinates to the mean trajectory $\bar{h}$.

At this point, we only need to determine the vector of variances of the samples V^{present} ,$V_{i,n}^{\text{future}}$, for $i = 1, 2, \ldots, N$, the center trusts γ^p and γ_i^f, and how the n_{paths} samples, at time n, are distributed across the past and the future belief sets. We compute the variance of the sample belief as how far (at each coordinate) the center is to be the mean of our predictions A_{n-1}^{present} or equivalently, "how big" was the correction we had to make in the previous step. We quantify this by computing the variance of the predictions around each center $V_n^{\text{present}} = V(A_{n-1}^{\text{pred}}, \alpha^{\text{present}})$ and $V_{i,n}^{\text{future}} = V(A_{n-1}^{\text{pred}}, \alpha_i^{\text{future}})$, for $i =, 1, 2, \ldots, N$, where the variance of a set $\{x_n\}_{i=1}^m$ centered at a vector x is almost the same as the variance of the set $\{x_n\}_{i=1}^m$ with x in place of the mean in the formula for the empirical variance. This means $V(\{x_n\}_{i=1}^m, x) = \frac{1}{m-1} \sum_{i=1}^m (x_i - x)^2$.

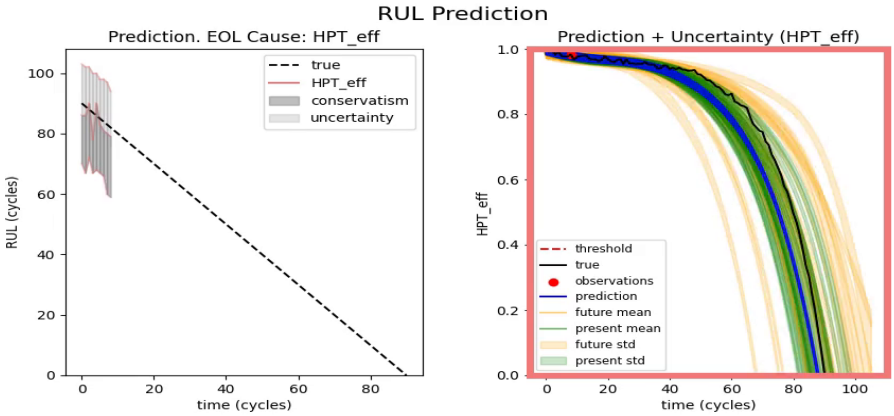


Figure 3 RUL prediction (on the left) and Performance prediction on the right on Unit 7 of N-CMAPSS-DS01 dataset. The green and yellow predicted distributions correspond to resulting distributions after the uncertainty propagation process with prior belief in A_n^p and A_n^f respectively. Tee blue distribution is the mean predicted distribution, the one we use for the RUL prediction.

We consider our trust in the respective beliefs as the inverse of the mean along components of the vector variance. This avoids the non-intuitive idea of dividing the belief by components of the linear combination. Remember our belief is on trajectories, and during the conversion from scalars to trajectories, many unrealistic trajectories may appear by just changing the sign of scalars. We also checked in practice that this gives us more stable predictions as variations in all dimensions are equal. Then, we define the center trusts as

$$\gamma_n^{\text{present}} = \frac{1}{\frac{1}{N}\sum_{j=1}^{N}[V_n^{\text{present}}]_j}, \qquad \gamma_{i,n}^{\text{future}} = \frac{1}{\frac{1}{N}\sum_{j=1}^{N}[V_{i,n}^{\text{future}}]_j}, \quad i = 1, 2, \ldots, N.$$

Finally, we chose the (present-future) sample rate as

$$r_n = \frac{\gamma_n^{\text{present}}}{\gamma_n^{\text{present}} + \frac{1}{N}\sum_{i=1}^{N}\gamma_{i,n}^{\text{future}}} \in [0, 1].$$

to give more beliefs to the centers we trust more. So, at time step n, the belief distribution is $|A_n^p| = \lfloor r_n n_{\text{paths}} \rfloor$ and $|A_n^f| = \lceil (1 - r_n) n_{\text{paths}} \rceil$ where $\lfloor x \rfloor$ denotes the floor function, $\lceil x \rceil$ denotes the ceiling function, and $|A|$ denotes the cardinality of the set A.

Uncertainty Propagation. Once we have built our belief set, we perform our Uncertainty Propagation step explained in the previous section, for the entire belief set. The α^* values obtained are the predictions at time step n. This means $A_n^{\text{pred}} = \tilde{A}_n^* = (A_n^{\text{p}})^* \cup (A_n^{\text{f}})^*$. Figure 3 on the right, shows the performance prediction obtained from the scalars in the prediction set A_n^{pred}, at a specific time step, differentiating the future and present belief predictions by color.

2.3.5 RUL and Uncertainty Computation

At each time t_n (starting by $t_0 = 0$) and for each of the κ performance metrics, we compute the prediction sets $A_{j,n}^{\text{pred}}$, for $j = 1, 2, \ldots, \kappa$, following the steps explained in Section 2.3.4 and we use these scalar sets to compute the set of n_{paths} prediction distributions

$$\text{Pred}_j(t_n) = \left\{ \sum_{i=1}^{n_{\text{paths}}} \alpha_i h_i : \alpha \in A_{j,n}^{\text{pred}} \right\}, \text{ for } j = 1, 2, \ldots, \kappa.$$

We obtain the predicted j-th performance metric at time step n, $\mathbf{f}_{j,n}$, by taking the average in both the mean and standard deviation of the predicted distributions $\text{Pred}_j(t_n)$. This means $\mathbf{f}_{j,n} \sim \mathcal{N}\left(\mu_{j,n}(t), \sigma^2_{j,n}(t)|t\right)$, where

$$(\mu_{j,n}(t), \sigma_{j,n}(t)) = \frac{1}{n_{\text{paths}}} \sum_{\mathcal{N}(\mu,\sigma)\in \text{Pred}_j(t_n)} (\mu(t), \sigma(t)).$$

Notice that we are taking the Gaussian distribution whose mean and standard deviation are the average of the mean and standard deviation of the predicted distributions, which is different from the average distributions, which would reduce the standard deviation by a factor of $\frac{1}{\sqrt{n_{\text{paths}}}}$. Assuming that the performance is decreasing, we compute the predicted EOL of the performance j at time t_n and its uncertainty boundaries as

$$\begin{aligned}
\text{EOL}_j(t_n) &= \min\{t \in [t_n, t_{\max}] : \mu_{j,n}(t) - 1.96\sigma_{j,n}(t) \leq 0\},\\
\text{EOL}_j^{\text{low}}(t_n) &= \min\{t \in [t_n, t_{\max}] : \mu(t) - 1.96\sigma(t) \leq 0, \mathcal{N}(\mu, \sigma^2) \in \text{Pred}_j(t_n)\},\\
\text{EOL}_j^{\text{up}}(t_n) &= \max\{t \in [t_n, t_{\max}] : \mu(t) + 1.96\sigma(t) \geq 0, \mathcal{N}(\mu, \sigma^2) \in \text{Pred}_j(t_n)\}.
\end{aligned}$$

Note that we are being conservative at choosing the lower bound of the 95% confidence interval $[\mu_{j,n}(t) - 1.96\sigma(t), \mu_{j,n}(t) + 1.96\sigma(t)]$. Finally, we predict the EOL and RUL of the system with its respective confident bounds at time t_n as follows

$$\begin{aligned}
\text{EOL}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j(t_n)\}, & \text{RUL}(t_n) &= \text{EOL}(t_n) - t_n,\\
\text{EOL}^{\text{low}}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j^{\text{low}}(t_n)\}, & \text{RUL}^{\text{low}}(t_n) &= \text{EOL}^{\text{low}}(t_n) - t_n,\\
\text{EOL}^{\text{up}}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j^{\text{up}}(t_n)\}, & \text{RUL}^{\text{up}}(t_n) &= \text{EOL}^{\text{up}}(t_n) - t_n.
\end{aligned}$$

3 Results

3.1 Dataset

We conducted experiments to validate our method on the N-CMAPSS dataset introduced in [1]. We used datasets 1 (N-CMAPSS-DS01) and 5 (N-CMAPSS-DS05). We selected N-CMAPSS-DS01 because it only had one failure mode and N-CMAPSS-DS05 as a more complex dataset with two failure modes. The N-CMAPSS dataset does not provide the performance metrics used to evaluate the health of the system, but following the instructions given in the paper, we can compute them by normalizing the health parameters with respect to the threshold using the formula $p(t) = 1 - \frac{\mu(t) - \mu_{\text{new}}}{\text{thr}}$, where μ denotes the health parameter, μ_{new} is the nominal value of the degradation parameter (with no degradation) and thr is the threshold. We approximate these two operating margins by taking the maximum and minimum along all the health parameter values of all the training units. In the case of N-CMAPSS-DS01, only the efficiency of the high-pressure turbine (HPT eff) is affecting the degradation of the system, while in dataset N-CMAPSS-DS05 we have the flow and efficiency high-pressure compressor (HPC flow and HPC eff). We selected one sample per cycle as a sample rate for training and testing.

3.2 MTL on N-CMAPSS dataset

We start by training our MTL model explained in Section 2.2 to learn the trajectory libraries (one for each performance metric). Each model was trained 10 times for each performance metric, changing weight initialization to account for model uncertainty. This is important

because, although all models should agree during the training period, the model uncertainty should increase as we learn beyond the training time. So, training several models and building an ensemble of them gives us a good estimation of the model uncertainty in the prolongation.

3.3 RUL prediction on the N-CMAPSS dataset

We evaluated our method on the four testing units in both datasets. To evaluate and compare our results, we used four evaluation metrics. Two for evaluating the prediction and the other two for the uncertainty bounds. To evaluate the prognostics' results, two commonly used metrics in prognostics analysis are the root-mean-square error (RMSE) and NASA's scoring function [16].

The latter metric penalizes over-predicting as this may cause serious damage to the system unlike an under-prediction error, which may only cause unnecessary extra maintenance of the system. To quantify the uncertainty on the RUL predictions, two widely used metrics are the Prediction Interval Coverage Probability (PICP) which quantifies the probability that the true RUL falls inside the corresponding prediction interval, and the prediction interval normalized width (PINAW), which represents the width of the prediction interval normalized on the range of actual RUL values.

We applied our method by taking the hyperparameters: $n_{\text{paths}} = 60$ and forgetting coefficient $w_{\max} = 1$, which implies almost no forgetting. Tables 1 summarize the results of our model (**LibRUL**) on both dataset N-CMAPSS-DS01 and N-CMAPSS-DS05, compared with some state-of-the-art uncertainty prediction models. The other model results have been taken from Tables 4 and 6 in [21]. The worst and best RUL prediction results on units of this dataset are shown in Figures 4a and 4b.

Table 1 Results for RUL prediction with uncertainty quantification in N-CMAPSS-DS01 and N-CMAPSS-DS05. Ours is LibRUL. Baseline results taken from [21].

	N-CMAPSS-DS01				N-CMAPSS-DS05			
Model	**RMSE**	**Score**	**PICP**	**PINAW**	**RMSE**	**Score**	**PICP**	**PINAW**
LibRUL	6.551	250.392	81.82%	**23.01%**	6.077	184.166	**91.74%**	29.08%
BGT	**4.285**	131.697	**96.77%**	27.16%	6.218	192.52	87.77%	**27.14%**
BLSTM	4.849	137.029	N/A	N/A	6.915	278.689	N/A	N/A
LSTM	5.396	195.57	N/A	N/A	7.522	263.393	N/A	N/A
DAT	4.57	**74**	N/A	N/A	**5.18**	**99**	N/A	N/A
DGP	7.03	14213.9	N/A	N/A	8	15499.2	N/A	N/A
GT	4.823	149.3	N/A	N/A	6.413	251.25	N/A	N/A

The results on the N-CMAPSS-DS01 dataset are slightly behind the state-of-the-art solutions. This is because two of the testing units, specifically units 7 and, especially, unit 9, fall outside the linear space $\mathcal{L}(h)$ generated by the library. These units exhibit a more accelerated degradation than any trajectory in the training dataset. To accurately capture this behavior, we would need to blend the training curves, modifying their curvature (through second-order transformations) to match these shapes. However, this level of curvature cannot be achieved through linear combinations of the library. The primary limitation of our approach is its inability to represent such curvature using linear combinations, which is a priority for future work and will be our immediate next step.

On the other hand, for N-CMAPSS-DS05, our model outperforms almost all state-of-the-art models (except for DAT) in both RMSE and Score, as shown in Table 1. This is expected, as DAT is a domain adaptation approach that uses significantly more data. Additionally, uncertainty quantification is not a focus of the DAT method. DAT employs Bayesian deep

learning techniques, specifically dropout layers, which provide only limited quantification of prediction uncertainty. Meanwhile, the BGT model is approximately 2% better than our method in terms of the uncertainty bound width of the predictions, but this comes at the cost of a 4% reduction in prediction interval coverage.

These results highlight the strengths of our model when the testing performance trajectories are "similar" to the training trajectories, especially when considering the increased explainability of our approach. Both BGT and DAT, while admitting uncertainty estimation, largely operate as "black-box" models. Although they effectively capture degradation patterns, they do not provide clear explanations for why specific predictions are made.

It is also important to note that our predictions are based solely on performance metrics, without leveraging feature information that could provide additional degradation insights. We also used a sampling rate of one cycle, which is equal to or greater than that of all competing approaches.

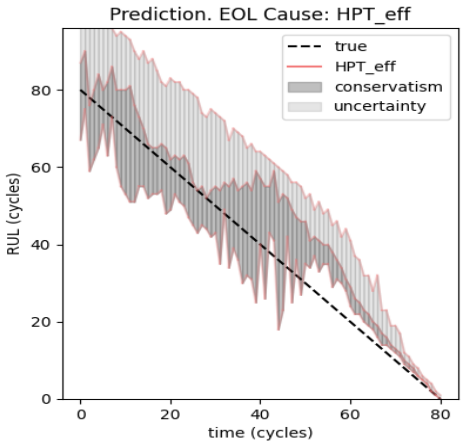
The worst and best RUL prediction results for units in this dataset are shown in Figures 4c and 4d. Note also that our predictions become very precise, with significantly less uncertainty toward the end of the lifecycle compared to state-of-the-art results. This is because our method balances present and future uncertainty as more observations are collected, and in the final stages, future uncertainty is significantly reduced. This is paramount in prognostics to avoid system failure. The only exception is in Figure 4c, where a precision delay occurs because one of the two performance metrics that drive the RUL estimation process remains very healthy, far from showing a clear degradation pattern and resulting in a larger uncertainty while the other performance metric suddenly decreases in performance.

An additional feature of our model is that it also predicts the cause of failure at each time step given by the performance that is predicted to cross the threshold earlier. Figures 4c and 4d show this in the color of the RUL predictions. Notice that in the case of 4c, the cause of failure is predicted correctly from a very early time step (around cycle 30).

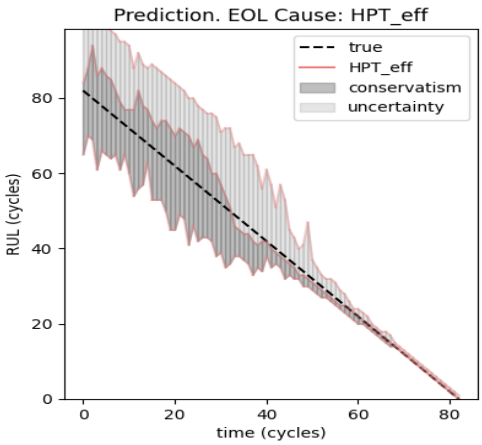
4 Conclusions

This paper presents a novel data-driven methodology for prognostics, inspired by the model-based RUL estimation approach. Our proposed approach bridges the gap between these two paradigms by developing a data-driven method that mimics the reasoning processes of model-based approaches. It leverages performance metrics and their associated thresholds to predict RUL in real-time, without requiring RUL labels, which are expensive and labor-intensive to acquire. Additionally, our model quantifies uncertainty in its predictions, offering valuable insights into which specific performance metric thresholds have been violated. This feature enhances the explainability of the RUL prediction process, making it more transparent and actionable for decision-makers.

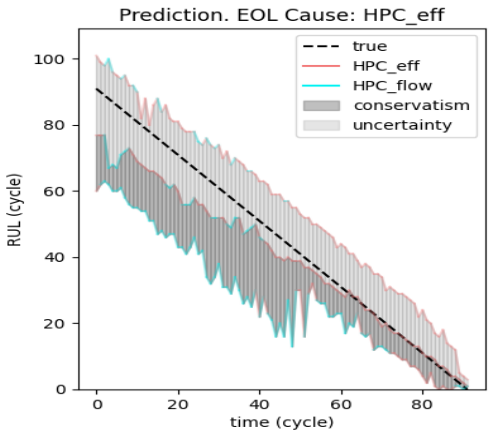
We empirically validated our approach using two datasets from the widely used N-CMAPSS benchmark. The results demonstrate the effectiveness of our methodology in predicting RUL while also providing confidence bounds for the predictions. For one of the datasets, the results are very close to the best state-of-the-art performance. However, in the other dataset, our results fall behind the current state of the art, highlighting a limitation of our approach when testing trajectories are not linearly representable by the training data. Our model struggles with datasets where the test data distribution significantly diverges from the training data. Nevertheless, the combination of explainability, uncertainty quantification, and online prediction makes our approach a promising alternative to both purely data-driven and traditional model-based methods, particularly in the context of complex systems. Future



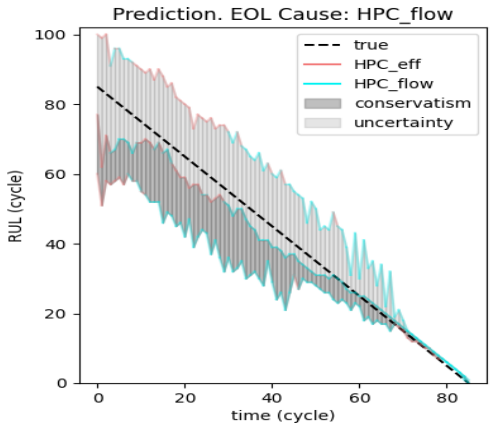
(a) RUL predictions on the C-MAPSS-DS01. Unit 9.



(b) RUL predictions on the C-MAPSS-DS01 Unit 10.



(c) RUL predictions on the C-MAPSS-DS05. Unit 8.



(d) RUL predictions on the C-MAPSS-DS05. Unit 7.

Figure 4 RUL predictions examples. Worst example on the left, best example on the right for each dataset. The color in the predictions indicates the performance that, at current time, the model predicts is going to cross the threshold first. The line in the middle represents the predicted value.

work will explore addressing the out-of-distribution issue by augmenting the model's ability to generalize beyond simple linear combinations of training data. Specifically, we are working on approaches to augment the performance model's predictions accounting for past data without constraining the model's inference space to a linear combination of past trajectories. This is especially important when predicting for degrading conditions which have not been experienced in the past, i.e. there is not representative data available for these operating conditions.

References

1 Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. *Data*, 6(1):5, 2021. doi:10.3390/DATA6010005.

2 Piero Baraldi, Francesca Mangili, and Enrico Zio. A prognostics approach to nuclear component degradation modeling based on gaussian process regression. *Progress in Nuclear Energy*, 78:141–154, 2015.

3 Yuanhong Chang, Fudong Li, Jinglong Chen, Yulang Liu, and Zipeng Li. Efficient temporal flow transformer accompanied with multi-head probsparse self-attention mechanism for remaining useful life prognostics. *Reliability Engineering & System Safety*, 226:108701, 2022. doi:10.1016/J.RESS.2022.108701.

4 Manuel Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Fusing physics-based and deep learning models for prognostics. *Reliability Engineering & System Safety*, 217:107961, 2022. doi:10.1016/J.RESS.2021.107961.

5 Matthew Daigle and Kai Goebel. Model-based prognostics with concurrent damage progression processes. *IEEE Transactions on Systems, man, and cybernetics*, 43(3):535–546, 2012. doi:10.1109/TSMCA.2012.2207109.

6 Matthew Daigle, Bhaskar Saha, and Kai Goebel. A comparison of filter-based approaches for model-based prognostics. In *IEEE Aerospace conference*, pages 1–10, 2012.

7 Austin Downey, Yu-Hui Lui, Chao Hu, Simon Laflamme, and Shan Hu. Physics-based prognostics of lithium-ion battery using non-linear least squares with dynamic bounds. *Reliability Engineering & System Safety*, 182:1–12, 2019. doi:10.1016/J.RESS.2018.09.018.

8 Jian Guo, Zhaojun Li, and Meiyan Li. A review on prognostics methods for engineering systems. *IEEE Transactions on Reliability*, 69(3):1110–1129, 2019. doi:10.1109/TR.2019.2957965.

9 Cheng-Geng Huang, Hong-Zhong Huang, and Yan-Feng Li. A bidirectional lstm prognostics method under multiple operational conditions. *IEEE Transactions on Industrial Electronics*, 66(11):8792–8802, 2019. doi:10.1109/TIE.2019.2891463.

10 Hamed Khorasgani, Gautam Biswas, and Shankar Sankararaman. Methodologies for system-level rul prediction. *Reliability Engineering & System Safety*, 154:8–18, 2016.

11 Jianfeng Lv, Zhongliang Yu, Haochuan Zhang, Guanghui Sun, Patrick Muhl, and Jianxing Liu. Transformer based long-term prognostics for dynamic operating pem fuel cells. *IEEE Transactions on Transportation Electrification*, 10(1):1747–1757, 2023.

12 Mohamed Marei and Weidong Li. Cutting tool prognostics enabled by hybrid cnn-lstm with transfer learning. *The International Journal of Advanced Manufacturing Technology*, 118(3):817–836, 2022.

13 Khanh TP Nguyen, Kamal Medjaher, and Christian Gogu. Probabilistic deep learning methodology for uncertainty quantification of remaining useful lifetime of multi-component systems. *Reliability Engineering & System Safety*, 222:108383, 2022. doi:10.1016/J.RESS.2022.108383.

14 Seol-Hyun Noh. Analysis of gradient vanishing of rnns and performance comparison. *Information*, 12(11):442, 2021. doi:10.3390/INFO12110442.

15 Darius V Roman, Ross W Dickie, David Flynn, and Valentin Robu. A review of the role of prognostics in predicting the remaining useful life of assets. In *27th European Safety and Reliability Conference 2017*, pages 897–904. CRC Press, 2017.

16 Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management*, pages 1–9. IEEE, 2008.

17 Zhigang Tian. Artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring. *Journal of intelligent Manufacturing*, 23:227–237, 2012. doi:10.1007/S10845-009-0356-9.

18 A Vaswani. Attention is all you need. *Advances in NeurIPS*, 2017.

19 Simon Vollert and Andreas Theissler. Challenges of machine learning-based rul prognosis: A review on nasa's c-mapss data set. In *2021 26th IEEE international conference on emerging technologies and factory automation (ETFA)*, pages 1–8. IEEE, 2021. doi:10.1109/ETFA45728.2021.9613682.

20 Ping Wang, Zhiqiang Long, and Gao Wang. A hybrid prognostics approach for estimating remaining useful life of wind turbine bearings. *Energy Reports*, 6:173–182, 2020.

21 Feifan Xiang, Yiming Zhang, Shuyou Zhang, Zili Wang, Lemiao Qiu, and Joo-Ho Choi. Bayesian gated-transformer model for risk-aware prediction of aero-engine remaining useful life. *Expert Systems with Applications*, 238:121859, 2024. doi:10.1016/J.ESWA.2023.121859.

Detecting Soft Faults in Heat Pumps

Birgit Hofer
Graz University of Technology, Austria

Franz Wotawa
Graz University of Technology, Austria

Abstract

Heat pumps are critical for energy-efficient heating and cooling, but their performance can be compromised by soft faults like condenser silting. It is vital to detect and fix such faults early in order to ensure optimal performance and longevity of heat pump systems, and consequently optimize the positive effect of heat pumps to our environment. In this paper, we tackle the problem of early fault detection and propose a supervised machine learning approach that detects soft faults. In particular, we used a random forest approach to learn the regular behavior of heat pumps. We detect faults via comparing the expected behavior obtained from the learned model with the current behavior. In addition to the description of the used methodology, we provide and discuss the results obtained from an experimental study that is based on synthetic data of two different heat pumps.

2012 ACM Subject Classification Hardware → Error detection and error correction

Keywords and phrases Fault detection, heat pumps, supervised machine learning

Digital Object Identifier 10.4230/OASIcs.DX.2024.22

Category Short Paper

Funding The research described in this paper has been carried out within the FFG AI for Green program as part of the project FO999899931 GreenHeat – The next generation of AI-based services for heat pumps funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK).

1 Introduction

Heat pumps are critical components in modern heating, ventilation, and air conditioning (HVAC) systems, providing efficient thermal regulation in residential and commercial buildings. However, the performance and longevity of heat pumps can be significantly compromised by soft faults. Soft faults refer to minor or intermittent issues that do not cause the system to fail immediately but can affect its performance or efficiency. Typical soft faults in the context of heat pumps are condenser silting and compressor efficiency loss [9]. These faults, while not immediately catastrophic, can lead to reduced efficiency, increased energy consumption, and eventual system failure if left undetected [10].

In the context of global efforts to achieve climate neutrality, maintaining the efficiency of heat pumps is more important than ever. Efficient HVAC systems play a crucial role in reducing greenhouse gas emissions and minimizing energy consumption [13]. Studies indicate that 15-30 % of a building's energy consumption is due to faults in HVAC systems or inefficient operations [7]. Moreover, even highly modern equipment such as heat pumps are very likely to operate at only 70-80 % efficiency due to a lack of maintenance activities. By detecting, diagnosing, and repairing faults in heat pumps early on, energy loss can be reduced by approximately 40 % [3].

Therefore, early detection and rectification of soft faults like condenser silting are essential to ensure that heat pumps operate at optimal efficiency, contributing to the broader goal of environmental sustainability. However, detecting soft faults in heat pumps, is challenging because these faults often cause subtle performance degradations that do not immediately


© Birgit Hofer and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



trigger alarms or noticeable system failures. The gradual nature of these faults means they can go unnoticed for extended periods, leading to inefficiencies and potential long-term damage. Additionally, the variability in operational conditions and the complexity of heat pump systems make it difficult to distinguish between normal fluctuations and actual faults.

Traditional fault detection methods often rely on manual inspections and reactive maintenance, which can be time-consuming and costly. Fault Detection and Diagnosis (FDD) [5, 15, 3] for heat pumps is a critical area of research aimed at enhancing the efficiency, reliability, and longevity of these systems. Heat pumps, which are pivotal in reducing reliance on fossil fuels and promoting sustainable energy use, can suffer from various faults that degrade performance and increase operational costs. FDD systems are designed to automatically detect anomalies and diagnose faults in real-time, ensuring timely maintenance and preventing severe breakdowns. By leveraging advanced techniques such as machine learning and data analytics, modern FDD systems can identify subtle patterns and deviations from normal operation, providing early warnings and detailed diagnostics.

In this paper, we propose an approach for detecting soft faults in air-to-water heat pumps using a supervised machine learning (ML) model. The approach leverages sensor data from a fault-free heat pump to train the model, enabling it to predict expected sensor values under normal operating conditions. By comparing the predicted sensor values with the actual sensor readings, the system can identify discrepancies indicative of potential faults. Specifically, when actual sensor values exceed the predicted values, it suggests the occurrence of soft faults. This proactive fault detection method aims to enhance the reliability and efficiency of heat pumps, reducing maintenance costs and preventing unexpected breakdowns.

The remainder of this paper is organized as follows: Section 2 reviews related work in the field of fault detection for HVAC systems. Section 3 details the methodology, including the model training, and the deployment phase. Section 4 presents the experimental results. Finally, Section 5 concludes the paper and describes future research directions.

2 Related work

Zhao and colleagues [16] provide an in-depth analysis of AI-based FDD methods for building energy systems. They explore the benefits, challenges, and future directions of integrating AI into FDD processes, emphasizing its potential to enhance energy efficiency and operational reliability in building management.

Nelson and Culp [11] examine the application of ML techniques in FDD methods of building systems. While Nelson and Culp [11] focus on supervised, unsupervised and reinforcement methods, Zhao and colleagues' [16] review encompasses a wide range of AI methodologies, including machine learning, expert systems, and other AI-driven approaches.

Chen and colleagues [6] review automated FDD approaches of HVAC systems. In particular, they focus on data-driven approaches (e.g., supervised and unsupervised learning, regression and statistics-based methods) and knowledge-based systems (e.g., physics-based modeling, Diagnostic Bayesian Network, and performance indicator-based methods).

Aguilera and colleagues [1] provide a comprehensive overview of the various faults that can occur in large-scale heat pump systems. The authors systematically categorize these faults, analyze their causes, and discuss their impacts on system performance and energy consumption. The paper categorizes faults into several types, including mechanical faults, electrical faults, thermal faults, and refrigerant-related faults. Common implications of these faults on the overall performance of heat pump systems are increased energy consumption, decreased overall system efficiency, and higher energy bills and maintenance costs; Severe

faults can result in system failures, leading to costly downtime and loss of productivity in industrial applications. The review highlights the need for proactive management of faults in large-scale heat pumps to ensure optimal performance and energy efficiency. By understanding the common faults, their causes, and impacts, stakeholders can implement better practices and technologies to mitigate these issues.

Bellanco and colleagues [3] describe common faults in heat pumps, their impact on system performance, and the methodologies employed for fault detection and diagnosis. Similar to [1], they categorize faults into mechanical, electrical, and refrigerant-related issues, highlighting how these faults can lead to decreased efficiency, increased energy consumption, and potential system damage. The authors discuss traditional measurement techniques and advanced detection methods, including the use of virtual sensors, which leverage data-driven approaches to estimate system parameters and identify anomalies in real-time. This paper provides support to determine (1) which faults can be tested under laboratory conditions or by simulation with a virtual model and (2) the features that identify those faults.

Bode and colleagues [4] present a case study in which trained ML models were applied to a real-world system in order to detect faults in heat pumps. Their results show that the trained models perform well on the synthetic dataset, but poorly on the real-world data set. The authors identify several reasons for the poor performance on the real data set, in particular data quality issues. Furthermore, they discuss other challenges such as the need for extensive training data, the complexity of model interpretation, the potential for overfitting and the need for ongoing model validation as system conditions change.

While Bode and colleagues use classification algorithms, we use regression algorithms. Bode and colleagues rely on labeled training data (labels: non-faulty and certain fault types) and the model learns to classify the sensor data into these fault categories. Our approach predicts the expected values of certain sensors based on the actual data from other sensors. The core idea here is to establish a baseline of normal operation. The advantage of Bode and colleagues' approach is that there is only one model required to detect different kinds of faults; our approach requires a model for each fault type. However, the advantage of our approach is that we do not need manually labeled data; instead we learn the normal behavior from the sensor data of a fault-free heat pump.

In a recent study, Barandier and colleagues [2] have compared four classification approaches for fault detection in heat pumps.

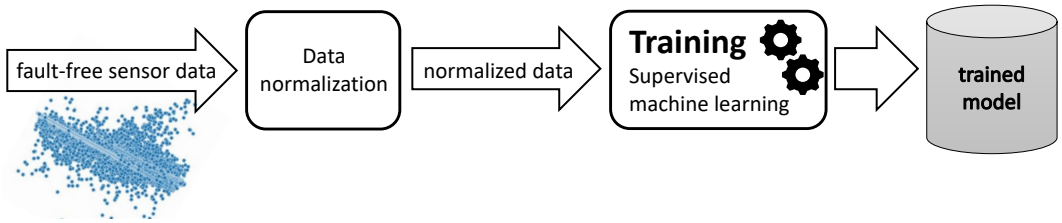
Borges and colleagues [5] focus on the fouling of the evaporator in air-to-water heat pumps. Fouling reduces heat transfer and therefore can cause performance degradation. They propose an FDD method based on artificial neural networks to detect this problem.

Nelson and Dieckert [12] discuss the strength and weaknesses of ML-based FDD approaches over rule-based approaches in commercial building systems. In particular, they mention the detection of insidious faults and the fault detection in cases where not all sensors are available as advantage of ML-based FDD approaches.

Khorasgani and Biswas [8] addressed the problem of incomplete knowledge in physics-based fault detection systems. They proposed a hybrid system that integrates data-driven and model-based methodologies to enhance the monitoring and management of smart buildings. The authors argue that traditional monitoring systems often rely solely on either empirical data or theoretical models, which can lead to inefficiencies and inaccuracies in performance assessment. The proposed framework utilizes real-time data collected from building sensors alongside predictive models that simulate building behavior under various conditions. This combination enables more accurate predictions of energy consumption, occupant comfort, and system performance, facilitating proactive maintenance and operational optimization.

Shin, Han and Rhee [14] investigated the interaction of AI-based fault detection methods for wind mills and human inspectors. They identified a positive effect of AI-based fault detection methods on the inspectors' specificity and time efficiency and concluded that AI-based fault detection methods are beneficial for predictive maintenance.

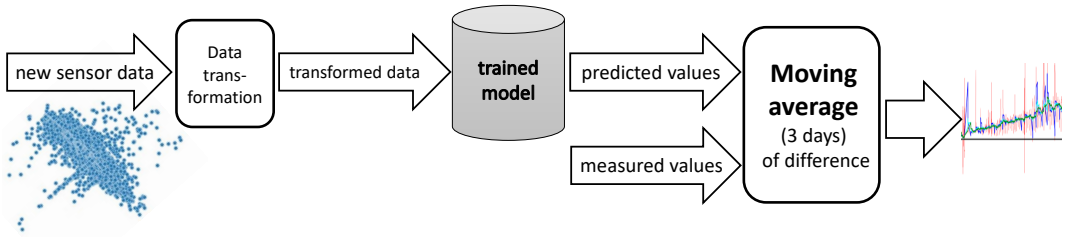
3 Method



(a) Training phase to learn the values the target sensor should have.

	Sensor	Description
Input features	wp-comp-speed	compressor speed
	pkp-power	heating output
	wp-app-state	operational status
	pkp-rlt	heating return temperature
	pkp-vlt	heating flow temperature after heat exchange
	pkp-flow	flow
	DT	compressor outlet temperature
	T5	fluid temperature before expansion valve
	T1	oil separator temperature
	wp-evap-t	evaporation temperature
Target variable	wp-cond-t	condensation temperature

(b) Sensor data that are used as input features and target value for detecting a silted condenser.



(c) Deployment phase.

Figure 1 Summary of the proposed method.

Figure 1 illustrates the applied method that consists of a training phase (see Figure 1a) and a deployment phase (see Figure 1c). During training, sensor data from a fault-free heat pump is used to develop a supervised machine learning regression model. Before training the model, the sensor data has be normalized. For this purpose, we use a standard scaler that subtracts from each sensor value x the sensor's mean μ and divides it by the sensor's standard deviation σ:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

The choice of sensors as features and the target variable depends on the specific fault type being detected (see Figure 1b). For instance, a silted condenser results in a higher condensation temperature; efficiency loss in the compressor can be noticeable through an increased power consumption of the inverter. Depending on the available sensor data and the problem that should be detected, different input features and target variables have to be defined. The set of relevant input features and the target variable illustrated in Figure 1b have been identified by domain experts: a heat pump manufacturer and a researcher from the Institute of Thermal Engineering at Graz University of Technology.

The regression models can be customized and trained individually for each heat pump after installation. This means that once a heat pump is installed, its unique operational characteristics and environmental conditions can be taken into account. By using the specific sensor data from the newly installed heat pump, the model can be fine-tuned to accurately reflect its normal operating conditions. This individualized training enhances the model's ability to detect deviations from the norm. Consequently, this approach ensures more precise fault detection tailored to each specific heat pump.

Once the model has been trained using the sensor data from a fault-free heat pump, it is ready for deployment. During the deployment phase (see Figure 1c), the model processes the incoming sensor data in real-time. In a first step, new sensor data is transformed accoriding to Eq. (1) using mean μ and standard deviation σ that where obtained in the training phase. Then, the transformed data is used to predict the expected values of the target sensor. The predicted sensor values are continuously compared to the actual sensor values. Since soft faults, such as condenser silting, typically develop slowly and because of fluctuations in the measured individual values, we compute a three-days average[1] of the differences between the measured and predicted values. By averaging over three days, we can smooth out short-term fluctuations and focus on persistent deviations that may indicate a fault. The continuous monitoring and averaging over several days help in identifying these gradual changes, ensuring that even slowly emerging issues are detected early.

Application in practice. In practice, sensor data can be very noisy. In particular, when starting the heat pump (e.g., after installation, maintenance work, or power interruptions), the sensors often return maximum or minimum values. Such obvious faulty sensor data has to be filtered out before training the model and predicting values. Furthermore, instead of constantly monitoring the deviation graphs, the monitoring system could raise an alarm if a certain threshold is exceeded for several days in a row.

4 Empirical evaluation

Data set. We use a benchmark of synthetic data that was created for the purpose of detecting silted condensers by a researcher from the Institute of Thermal Engineering using the Transient System Simulation Tool (TRNSYS)[2]. The benchmark consists of four files with (fault-free) training data for two air-to-water heat pumps (files A1 and A2 for sensor data of heat pump A, and files C1 and C2 for sensor data of heat pump C), and eight files with test data for the silting problem (four files per heat pump type). In half of the test files the actual problem occurs at some point in time, the other half is fault-free. Each file contains the minute-by-minute sensor data for one year, i.e. 525 600 samples (60x24x365) per file.

[1] The computation of the three-days average considers only the data of those time stamps where the heat pump is actually running.

[2] `https://www.trnsys.com/`

Table 1 RMSE of the models for condenser silting on the 25 % test set from the training files.

Model / Data set	A1	A2	A1+A2	C1	C2	C1+C2
LinearSVR	0.305	0.229	0.251	0.483	0.480	0.509
RandomForestRegressor	**0.082**	**0.061**	**0.052**	**0.095**	**0.087**	**0.072**
LinearRegression	0.244	0.190	0.219	0.436	0.441	0.456
GradientBoostingRegressor	0.130	0.119	0.125	0.174	0.180	0.180
SGDRegressor	0.247	0.196	0.219	0.476	0.470	0.476
Lasso	1.114	1.084	1.087	1.226	1.236	1.246
KNeighborsRegressor	**0.074**	**0.078**	**0.061**	**0.083**	**0.081**	**0.068**

Training phase. We implemented our approach in Python 3 and used scikit-learn (version 1.4.2) as ML library. In the training phase, we created different regression models: LinearSVR, RandomForestRegressor, LinerRegression, GradientBoostingRegressor, SGDRegressor, Lasso, and KNeighborsRegressor. We chose these models because they are capable to handle datasets with $> 500K$ samples and 10 input features. In order to decide which models are best suited for our purpose, we randomly spilt the training files into a 75 % training set and a 25 % test set and we used the square root of the mean squared error (RMSE) as decision criterion. We trained the models on the individual training files (A1, A2, C1, and C2) and on their combinations (A1+A2, C1+C2). Table 1 shows the results for the condenser silting models. For all six input data sets, the RandomForestRegressor and the KNeighborsRegressor perform best. Therefore, we will use these models in the deployment phase.

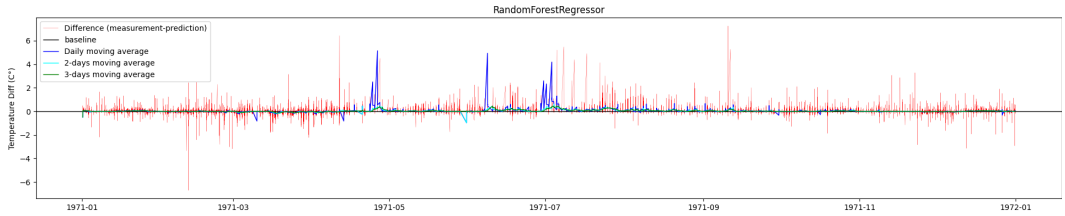
We omitted the feature importance analysis because the features were engineered by domain experts, ensuring their relevance and significance. We did not perform hyperparameter tuning because the RandomForestRegressor and KNeighborsRegressor are already performing well and the incremental improvements might not justify the additional time and resources and we wanted to minimize the risk of overfitting. The chosen baseline model with default hyperparameters are sufficient for the task at hand. Therefore, further tuning is unnecessary.

Deployment phase. For this part of the evaluation, we show the results of the RandomForestRegressor models that were trained on both training files for each heat pump type (i.e. A1+A2 and C1+C2). Figure 2 illustrates the results for the condenser silting test files for heat pump A. The red lines indicate the difference of the actual sensor values and the predicted values for the individual minute-by-minute time stamps; the blue, cyan, and green lines show the one-, two-, and three-days moving averages of these differences. For test file 1 (Figure 2a) and test file 3 (Figure 2c), no silting occurs during the year. The condenser in test file 2 (Figure 2b) got silted in April and the condenser in test file 4 (Figure 2d) got silted in June.

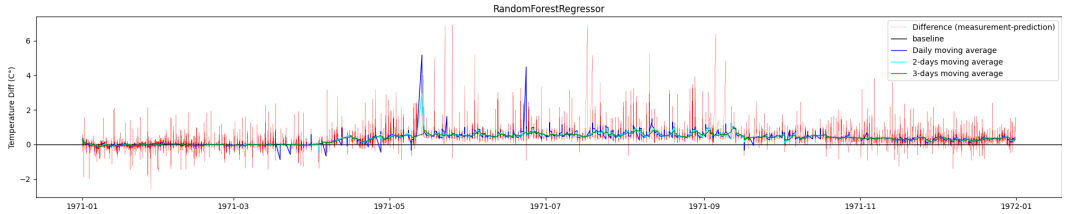
Figure 3 shows the results for heat pump C. Again, there a two files without any silting (Figures 3a and 3c) and two files with silting, one starting in March (Figure 3b) and one starting in September (Figure 3d).

In conclusion, the empirical evaluation has demonstrated the effectiveness of the proposed approach for detecting soft faults, such as condenser silting, in heat pumps.

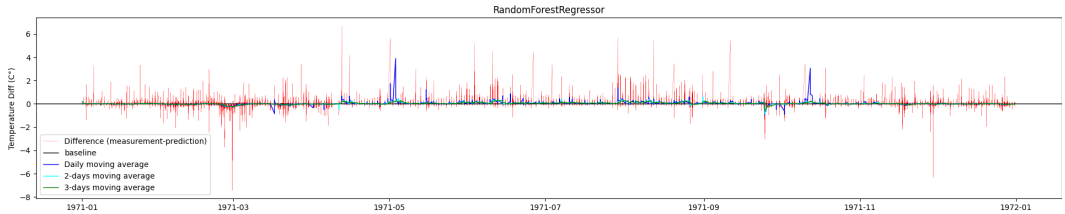
Threats to validity. The biggest threat to validity concerns the external validity. The evaluation was performed on synthetic data only. While the data set was created using realistic weather models, the findings might not generalize to real-world settings with varying



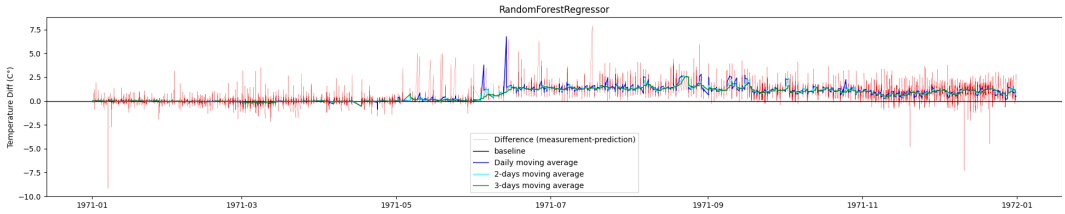
(a) Test 1: No condenser silting.



(b) Test 2: Condenser silting starting in April.



(c) Test 3: No condenser silting.



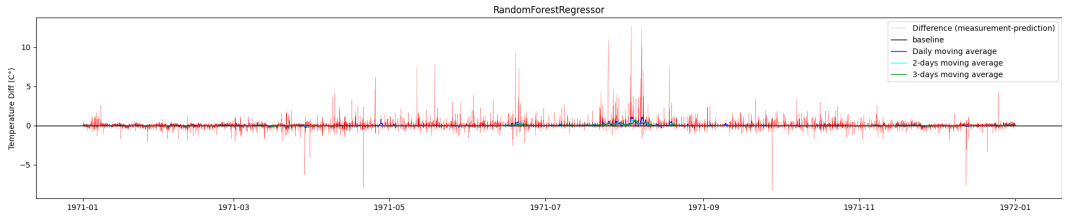
(d) Test 4: Condenser silting starting in June.

Figure 2 Condenser silting heat pump A.

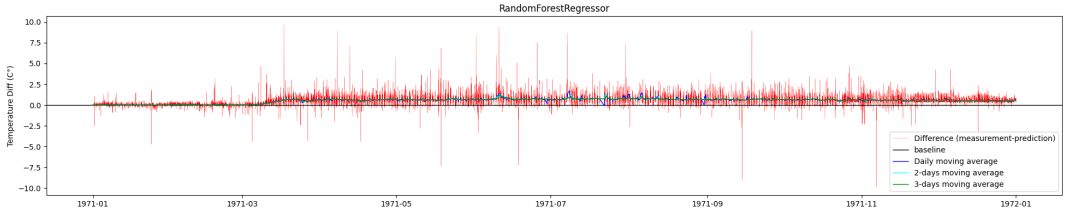
environmental conditions. Furthermore, the accuracy and completeness of the sensor data used for training and testing the model are crucial. Any noise, missing values, or biases in the data can lead to incorrect fault detection. It is up to future work to investigate our approach on real-world sensor data.

5 Conclusion and future work

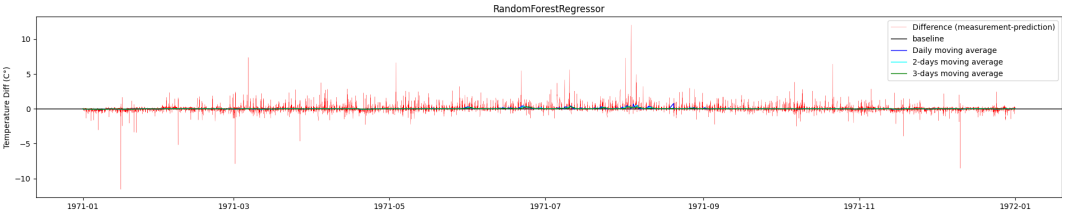
The empirical evaluation has demonstrated the effectiveness of the proposed approach for detecting condenser silting in air-to-water heat pumps. By leveraging a supervised learning model trained on fault-free sensor data, the method accurately predicts expected sensor values and identifies discrepancies indicative of potential faults. The results confirm that this approach not only reliably detects condenser silting but also enhances the overall maintenance strategy for heat pumps. This contributes to improved operational efficiency, reduced downtime, and extended equipment lifespan.



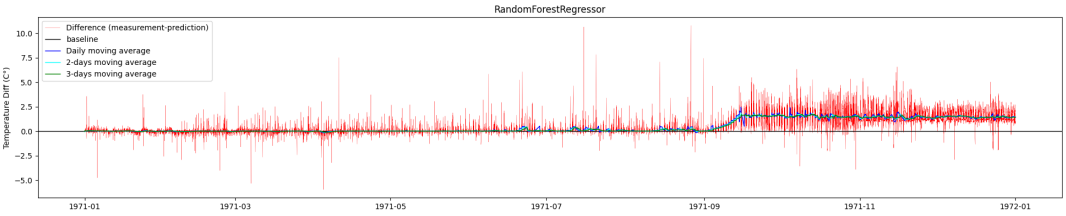
(a) Test 1: No condenser silting.



(b) Test 2: Condenser silting starting in March.



(c) Test 3: No condenser silting.



(d) Test 4: Condenser silting starting in September.

Figure 3 Condenser silting heat pump C.

However, the proposed approach has some limitations. Firstly, the accuracy of fault detection heavily depends on the quality and quantity of the training data. If the training data does not adequately represent all possible operating conditions, the model may not perform well in real-world scenarios. Secondly, the approach assumes that the fault-free data is accurate, which might not always be the case. Any noise or errors in the training data can lead to incorrect predictions. Despite these limitations, the approach provides a valuable tool for early fault detection and maintenance planning.

In future work, we will work with real heat pump data. The effectiveness of AI algorithms heavily depends on the quality and quantity of data available for training. Inconsistent or insufficient data can lead to inaccurate predictions [16, 11]. We will explore techniques to filter out wrong sensor data beforehand and explore the impact of wrong sensor data on the results. Furthermore, we will investigate how much training data is actually required to train the models. So far, we have used at least one year of data for training the model, but in practice such long training periods are infeasible. In the presented evaluation, we

have focused on the detection of condenser silting. In future work, we will evaluate the detection of other soft faults such as the efficiency loss of the compressor and we will explore the influence of the presence of multiple fault on the results.

References

1 José Joaquín Aguilera, Wiebke Meesenburg, Torben Ommen, Wiebke Brix Markussen, Jonas Lundsted Poulsen, Benjamin Zühlsdorf, and Brian Elmegaard. A review of common faults in large-scale heat pumps. *Renewable and Sustainable Energy Reviews*, 168:112826, October 2022. doi:10.1016/j.rser.2022.112826.

2 Pedro Barandier, Mateus Mendes, and Antonio J. Marques Cardoso. Comparative analysis of four classification algorithms for fault detection of heat pumps. *Energy and Buildings*, 316:114342, August 2024. doi:10.1016/j.enbuild.2024.114342.

3 Ivan Bellanco, Elena Fuentes, Manel Vallès, and Jaume Salom. A review of the fault behavior of heat pumps and measurements, detection and diagnosis methods including virtual sensors. *Journal of Building Engineering*, 39:102254, July 2021. Publisher: Elsevier. doi:10.1016/J.JOBE.2021.102254.

4 Gerrit Bode, Simon Thul, Marc Baranski, and Dirk Müller. Real-world application of machine-learning-based fault detection trained with experimental data. *Energy*, 198:117323, May 2020. doi:10.1016/j.energy.2020.117323.

5 Sebastian Borges, Lasse Jöhnk, Tim Klebig, Christian Vering, and Dirk Müller. Fault Detection and Diagnosis by Machine Learning Methods in Air-To-Water Heat Pumps: Evaluation of Evaporator Fouling. In *36th International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems (ECOS 2023)*, 2023. doi:10.52202/069564-0074.

6 Jianli Chen, Liang Zhang, Yanfei Li, Yifu Shi, Xinghua Gao, and Yuqing Hu. A review of computing-based automated fault detection and diagnosis of heating, ventilation and air conditioning systems. *Renewable and Sustainable Energy Reviews*, 161:112395, June 2022. Publisher: Pergamon. doi:10.1016/J.RSER.2022.112395.

7 Zhelun Chen, Zheng O'Neill, Jin Wen, Ojas Pradhan, Tao Yang, Xing Lu, Guanjing Lin, Shohei Miyata, Seungjae Lee, Chou Shen, Roberto Chiosa, Marco Savino Piscitelli, Alfonso Capozzoli, Franz Hengel, Alexander Kührer, Marco Pritoni, Wei Liu, John Clauß, Yimin Chen, and Terry Herr. A review of data-driven fault detection and diagnostics for building HVAC systems. *Applied Energy*, 339:121030, 2023. doi:10.1016/j.apenergy.2023.121030.

8 Hamed Khorasgani and Gautam Biswas. A combined model-based and data-driven approach for monitoring smart buildings. In *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 21–36, 2018. doi:10.29007/g44l.

9 Necati Kocyigit, Huseyin Bulgurcu, and Cheng-Xian Lin. Fault diagnosis of a vapor compression refrigeration system with hermetic reciprocating compressor based on p-h diagram. *International Journal of Refrigeration*, 45:44–54, 2014. doi:10.1016/j.ijrefrig.2014.05.027.

10 Hatef Madani. Smart fault detection and diagnosis for heat pump systems. In *24th IIR International Congress of Refrigeration*, 2015. doi:10.18462/iir.icr.2015.0108.

11 William Nelson and Charles Culp. Machine Learning Methods for Automated Fault Detection and Diagnostics in Building Systems—A Review. *Energies 2022*, 15(15):5534, July 2022. Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/EN15155534.

12 William Nelson and Christopher Dieckert. Machine Learning-Based Automated Fault Detection and Diagnostics in Building Systems. *Energies*, 17(2):529, January 2024. Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/en17020529.

13 Selvapriya R, Gopinath M, Velmurugan D, and Tamilchelvan P. Climate Change Mitigation Through AI Solutions. In *Global Challenges for the Environment and Climate Change*, pages 252–277. IGI Global, 2024. doi:10.4018/979-8-3693-2845-3.ch014.

14 Won Shin, Jeongyun Han, and Wonjong Rhee. AI-assistance for predictive maintenance of renewable energy systems. *Energy*, 221:119775, April 2021. doi:10.1016/j.energy.2021.119775.

15 Yang Song, Davide Rolando, Javier Marchante Avellaneda, Gerhard Zucker, and Hatef Madani. Data-driven soft sensors targeting heat pump systems. *Energy Conversion and Management*, 279:116769, March 2023. doi:10.1016/j.enconman.2023.116769.

16 Yang Zhao, Tingting Li, Xuejun Zhang, and Chaobo Zhang. Artificial intelligence-based fault detection and diagnosis methods for building energy systems: Advantages, challenges and the future. *Renewable and Sustainable Energy Reviews*, 109:85–101, July 2019. Publisher: Pergamon. doi:10.1016/J.RSER.2019.04.021.

Diagnosing Non-Intermittent Anomalies in Reinforcement Learning Policy Executions

Avraham Natan
Ben-Gurion University of the Negev, Beersheba, Israel

Roni Stern
Ben-Gurion University of the Negev, Beersheba, Israel

Meir Kalech
Ben-Gurion University of the Negev, Beersheba, Israel

Abstract

Reinforcement learning (RL) algorithms output policies specifying which action an agent should take in a given state. However, faults can sometimes arise during policy execution due to internal faults in the agent. As a result, actions may have unexpected effects. In this work, we aim to diagnose such faults and infer their root cause. We consider two types of diagnosis problems. In the first, which we call RLDXw, we assume we only know what a normal execution looks like. In the second, called RLDXs, we assume we have models for the faulty behavior of a component, which we call fault modes. The solution to RLDXw is a time step at which a fault occurred for the first time. The solution to RLDXs is more informative, represented as a fault mode according to which the RL task was executed. Solving those problems is useful in practice to facilitate efficient repair of faulty agents, since it can focus the repair efforts on specific actions. We formally define RLDXw and RLDXs and design two algorithms called WFMa and SFMa for solving them. We evaluate our algorithms on a benchmark of RL domains and discuss their strengths and limitations. When the number of the observed states increases, both WFMa and SFMa report a decrease in runtime (up to significantly 6.5 times faster). Additionally, the runtime of SFMa increases linearly with the increase in candidate fault modes.

2012 ACM Subject Classification Hardware → Bug detection, localization and diagnosis; Theory of computation → Reinforcement learning

Keywords and phrases Diagnosis, Reinforcement Learning, Autonomous Systems

Digital Object Identifier 10.4230/OASIcs.DX.2024.23

Category Short Paper

Funding *Roni Stern*: Israeli Science Foundation (ISF) grant #1238/23

1 Introduction

As the use of autonomous systems increases, there is a need for approaches to address challenges in different aspects of the agent's task execution, such as planning and fault diagnosis. In this work, we ask what made an agent diverge from its expected course of action. We focus on a setting where the environment is dynamic and sometimes even unknown. In that case, the expected actions of an agent are defined by the so-called *Policy*, which is the product of *Reinforcement Learning* based planning approaches.

Reinforcement Learning (RL) is a technique for guiding an agent through a dynamic environment such that the agent maximizes the rewards it collects. To achieve this, RL models train on observation data available through simulations or actual interaction with the environment. The output of the RL model is referred to as *policy*. The policy guides the agent on what action to perform in every state until the agent's goal is reached.


© Avraham Natan, Roni Stern, and Meir Kalech;
licensed under Creative Commons License CC-BY 4.0



Sometimes, internal faults within the agent may occur. Such faults may change the agent's ability to perform its actions normally. This, in turn, may lead to some actions having unexpected effects. When this happens, we want to understand what caused the agent to deviate from its policy, that is a *Diagnosis*. Inferring the root cause can help repair the components responsible for executing faulty actions faster. In a scenario where repair is not possible, this can improve future planning if the faulty components can be assumed faulty, therefore computing suitable policies.

Diagnosis outlines a range of AI techniques that try to understand the root causes of failures in systems. Given a system, its expected behavior, and an observation, the diagnosis problem tries to infer what caused the observation to be different than expected. Previous work asks this question in a variety of settings such as Software engineering [1], Multi-Agent Systems [10], and more, and proposes different approaches such as Spectrum-based Fault Localization [1] and Model-Based Diagnosis (MBD) [15]. One difference between works in MBD is what can be assumed about the behavior mode of faulty system components. One approach assumes no knowledge on the faulty behaviour mode and is called Weak Fault Model (WFM) [3]. The other approach assumes a number of modes in which a faulty system component can be and is called Strong Fault Model (SFM) [21].

In this work, we solve the problem of diagnosing faulty RL policy executions. We use MBD concepts to define the problem for WFM (RLDXw), where the only assumption about the actions is whether they are faulty or healthy, and for SFM (RLDXs), where we assume a number of candidate fault modes that can explain the faulty execution, and where one of them is the correct one. In addition, we assume that non-intermittent faults are present in this work. This means that if an action of some type (for example, left acceleration) is faulty, then every time this action type is executed, it will execute in a faulty manner. Moreover, we assume that faulty actions behave in a single faulty manner, i.e., if the left acceleration results in a right acceleration, then whenever a left acceleration is attempted, the result will always be a right acceleration. The contributions of this work are as follows:

1. We define formally the problems RLDXw and RLDXs.
2. We propose two algorithms, WFMa and SFMa to solve RLDXw and RLDXs, respectively.
3. We evaluate the algorithms theoretically and empirically and discuss their strengths and weaknesses.

In particular, we ask the following two questions:

Q1 How does the number of candidate fault modes impact the performance of SFMa?
Q2 How does the number of visible states affect the performance of WFMa and SFMa?

We test WFMa and SFMa on the domains *Acrobot*, *CartPole*, *MountainCar* and *Taxi*, all of which can be found in the Gymnasium website[1]. We found out that the runtime of SFMa increases linearly with the increase in the number of candidate fault modes, and that the runtime of both algorithms increases with the decrease in percentage of observed states.

2 Background and Related Work

2.1 Markov Decision Problem

A *Markov Decision Problem* (MDP) is a sequential decision-making problem in which the cost and transition functions depend only on the current state of the system and the current action [14]. Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, P_a(S, S'), R_a(S, S') \rangle$ where $\mathcal{S}$ is a set of environment and agent states, $\mathcal{A}$ is the set of actions an agent can perform,

[1] `https://gymnasium.farama.org/`

$P_a(S, S') = Pr(S_{t+1} = S' | S_t = S, a_t = a)$ is the probability of reaching state S' given that the current state at time t is S and the agent executes action a. This is also known as the transition function and $R_a(S, S')$ is the immediate reward gained by executing action a from S.

A solution to an MDP is a *policy* that guides the agent to choose actions that maximize its cumulative reward. In this work, we consider deterministic policies. This means a policy π can be defined as a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a well-studied branch of Machine Learning designed for solving MDPs and other sequential decision-making problems [22]. Some RL algorithms, such as Deep Q-Networks (DQN), learn a function that estimates the value of performing actions in states [9]. The policy is computed implicitly by choosing in every state the action that maximizes this value function. Other RL algorithms, such as Asynchronous Advantage Actor-Critic [8] and Proximal Policy Optimization (PPO) [20], learn a policy directly.

The agent applies its actions, based on the learned policy or value function, on an *environment* that defines what the outcomes of those actions are. The environment defines the set of states an agent can be in and the transition function between the states. When an agent in state S executes action a, its next state is determined by the environment, which is specified by the probability $P_a(S, S')$ that is defined for the environment.

2.3 Automated Diagnosis

Automated diagnosis (DX) delves into the question of finding the root cause of a system failure. There are many approaches to formalize a diagnosis problem, and they are all essentially similar in the way they generally define a problem – when a system is not behaving the way it is supposed to behave, we say there is a diagnosis problem [15]. Diagnosis methods take two distinct approaches when modeling the system in question. The first approach is called *Weak fault model* (WFM). In WFM, there is no assumption about what is the behaviour of faulty parts in the system [3]. The parts are either assumed normal or not normal. Although simple, this approach provides diagnoses that are not very informative, As it can not provide further information about the behaviour of faulty parts. To complement this approach, *Strong fault model* (SFM) assumes known information about the faulty operation modes of the system parts [21].

In this work, we address faulty executions of actions in RL environments. Related work focused on diagnosing plan steps that were executed abnormally [18, 7, 10] while others try to diagnose faulty plan executions that happen due to disagreements between the agents' beliefs [17, 2, 5]. Some works use a single observation [4], while others use many [23]. More recent work uses approaches from software diagnosis to diagnose multi-agent systems [11, 12]. Recent surveys survey other advancements in diagnosis [6, 16].

3 Problem Definition

In this section, we define our problem. Let Π be an MDP and π a policy created for it, e.g., using some RL algorithm. For simplicity, we assume that the transition function in Π is deterministic. This allows us to define a *simulator* for Π as a function $\chi : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ such that $\chi(S, a)$ is the state the agent is expected to reach when executing a in state S. We extend the definition of χ to address the normal execution of π for i steps starting from S,

denoted as $\chi(S, \pi, i)$. This extension is defined as a function $\chi : \mathcal{S} \times \mathcal{A} \times \mathbb{N} \to \mathcal{T}$ that given a state, an action, and a number of steps returns a trajectory T, where the trajectory is an alternating sequence of actions and states, simulated by the normal execution of the policy π starting from state S.

In this work, we assume executing an action may fail, and there are several *fault modes* that specify different ways in which an action behaves when faulty.

▶ **Definition 1** (Fault Mode). *A fault mode is a function $f : \mathcal{A} \to \mathcal{A}'$ that takes an action $a \in \mathcal{A}$ and returns an action a' from a potentially different set of actions $\mathcal{A}'$. If $\forall a \in \mathcal{A}$ it holds that $f(a) = a$, we say that f is the healthy fault mode, and we denote it as h.*

In this work, we make the common single-fault assumption [19, 13], i.e., we assume that only one fault mode is present at any given time. However, a fault mode may affect multiple actions.

We extend the definition of χ to include simulating a policy together with a fault mode. That means that for a fault mode f, the simulator $\chi(S_0, \pi, i, f)$ outputs a trajectory $T = (S_0, \hat{a}_1, \hat{S}_1, ..., \hat{a}_i, \hat{S}_i)$ that corresponds to executing the policy from the state S_0 for i steps, where the faulty actions according to f fail. To support the normal definition defined previously as $\chi(S_0, \pi, i)$, we define it now as $\chi(S_0, \pi, i, h)$, where $h : \mathcal{A} \to \mathcal{A}$ is the model of the normal behavior, and is defined as $h(a) = a$.

▶ **Example 2.** Consider the *MountainCar* environment, a standard RL domain [24]. Figure 1 provides a visual representation. In this environment, a car is located between two hills. The car can only accelerate to the left, perform no action, and accelerate to the right. We denote these actions as L, N and R. The task of the car is to climb the right hill by strategically accelerating while using gravity to build its speed. Let us assume that in a normal execution,

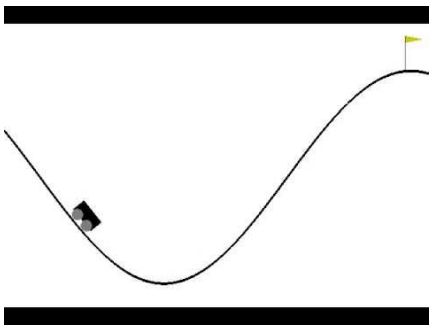


Figure 1 The MountainCar environment.

the car would reach its goal and the corresponding trajectory is $(a_1 = R, S_1, a_2 = R, S_2, a_3 = R, S_3, a_4 = N, S_4, a_5 = N, S_5, a_6 = L, S_6, a_7 = L, S_7, a_8 = L, S_8, a_9 = R, S_9, a_{10} = R, S_{10})$. In addition there are 3 candidate fault modes f_1, f_2, f_3 defined as:

- $\mathbf{f_1}(\mathbf{L}) = \mathbf{N}, f_1(N) = N, f_1(R) = R$
- $f_2(L) = L, f_2(N) = N, \mathbf{f_2}(\mathbf{R}) = \mathbf{N}$
- $\mathbf{f_3}(\mathbf{L}) = \mathbf{R}, f_3(N) = N, f_3(R) = R$

Assume the car is faulty and behaves according to fault mode f_1. As a result, actions a_6, a_7, and a_8 failed, and the car did not reach its goal. Also, suppose the car observed the states $\mathcal{O} = (S_0, S_5, S_7, S_{10})$. Because of the failed actions, the states $\hat{S}_7$ and $\hat{S}_{10}$ from the trajectory, returned by simulating $\chi(S_0, \pi, i, h)$, do not match the observed states S_7 and S_{10}.

In this work we aim to find the cause of the inconsistency between the expected healthy execution and the observed faulty execution of a policy π. We define this problem for WFM and for SFM separately. We begin by formally defining the problem for WFM, which we call RLDXw:

▶ **Definition 3** (RLDXw problem). *An RLDXw problem is defined as a tuple $\langle \mathcal{O}, \chi, \pi \rangle$ where π is a policy, χ is a simulator and $\mathcal{O} = (S_0, S_1, \ldots, S_n)$ is an observed series of states. An RLDXw problem arises when there exists an observed state S_i s.t. $S_i \neq \hat{S}_i$ where $\hat{S}_i$ is the state at step i of the trajectory generated by $\chi(S_0, \pi, i, h)$.*

A solution to an RLDXw problem is a *diagnosis*, which in the WFM setting is a time-step i, where the first fault occurred. Formally:

▶ **Definition 4** (RLDXw Diagnosis). *Given an RLDXw problem $\langle \mathcal{O}, \chi, \pi \rangle$, a diagnosis is a time-step i such that $\chi(S_0, \pi, i, h) \neq S_i$ and $\forall i' < i$ it holds that $S_{i'} \in \mathcal{O} \Rightarrow \chi(S_0, \pi, i', h) = S_{i'}$.*

Since RLDXw assumes WFM, it only knows how the effects of actions that were executed normally. Because of that, when two states $S_i, S_{i'}$ are observed, such that S_i is expected, but $S_{i'}$ is not, we can not reconstruct the actual execution. Alternatively, we define the problem for SFW, where we assume several candidate fault modes, out of which one is correct. We call this problem RLDXs. Formally:

▶ **Definition 5** (RLDXs problem). *An RLDXs problem is defined as a tuple $\langle \mathcal{O}, \chi, \pi, \mathcal{F} \rangle$ where π is a policy, χ is a simulator, $\mathcal{O} = (S_0, S_1, \ldots, S_n)$ is an observed series of states, and $\mathcal{F}$ is a set of fault modes. An RLDXs problem arises when there exists an observed state S_i s.t. $S_i \neq \hat{S}_i$ where $\hat{S}_i$ is the state at step i of the trajectory generated by $\chi(S_0, \pi, i, h)$.*

A solution to an RLDXs problem is a *diagnosis*, which in the SFM setting is a fault mode f for which there exists a trajectory generated by the execution of $\chi(S_0, \pi, i, f)$ that is consistent with the observations. Formally:

▶ **Definition 6** (RLDXs Diagnosis). *Given an RLDXs problem $\langle \mathcal{O}, \chi, \pi, \mathcal{F} \rangle$, a diagnosis is a fault mode f_j such that executing $\chi(S_0, \pi, n, f_j)$ can result in a trajectory $T = (S_0, \hat{a}_1, \hat{S}_1, ..., \hat{a}_n, \hat{S}_n)$ where for each observed state $S_i \in \mathcal{O}$ it holds that $S_i = \hat{S}_i$.*

4 Method Description

Next, we present the algorithms for solving the RLDXw and RLDXs.

4.1 Solving RLDXw

We present the algorithm called WFMa which stands for "Weak", since it assumes WFM. It shown in Algorithm 1. WFMa keeps track of the last index where a simulated state matched an observed state as i (lines 1,8), and when it first encounters mismatching states at step i', it returns a set of diagnoses $\mathcal{D} = \{i, i+1, ..., i'\}$ indicating the possible time steps at which an action could have failed for the first time.

▶ **Example 7.** Refer to Example 2. WFMa starts by keeping track of index 0, the index of the first state, as the variable i. It keeps looping throughout the states, updating i every time an observation is available. When WFMa compares the observed state S_7 with the simulated state $\hat{S}_7$, the states are not the same. At this stage, i has the value 5, which is the value of the last matching state. The algorithm will return the diagnoses $\mathcal{D} = \{5, 6, 7\}$, which are all possible diagnoses. This is because WFMa can not determine which of the 3 time steps is the time step of the fault's first occurence.

Algorithm 1 WFMa.

Input: π - The policy.
Input: χ - The simulator.
Input: $\mathcal{O} = (S_0, ..., S_n)$ - The observed states.
Result: $\mathcal{D} = \{d_1, d_2, ...\}$ - The set of diagnoses.

```
1  i ← 0
2  Ŝ_0 ← S_0
3  for i' = 1 to n do
4  |   â_{i'} ← π(Ŝ_{i'-1})
5  |   Ŝ_{i'} ← χ(Ŝ_{i'-1}, â_{i'})
6  |   if S_{i'} ∈ O then
7  |   |   if S_{i'} = Ŝ_{i'} then
8  |   |   |   i ← i'
9  |   |   else
10 |   |   |   return D = {i, i+1, ..., i'}
```

4.2 Solving RLDXs

To address this setting, we present the algorithm we call SFMa, which is detailed in Algorithm 2. The input to SFMa is the policy π, the simulator χ, the observation $\mathcal{O}$, and also a set of candidate fault modes $\mathcal{F}$. It returns a set of diagnoses $\mathcal{D} = \{d_1, d_2, ...\}$ where each diagnosis is a fault mode that is consistent with the observed states.

SFMa starts by creating an empty trajectory that includes S_0 for each of the k candidate fault modes (Lines 1-4). Then, it goes over the n steps in $\mathcal{O}$ until the candidate set of fault modes becomes size 1 (Lines 5,14). In each step and for each fault mode, SFMa simulates the next state $\hat{S}_{i,j}$ based on the previous state $\hat{S}_{i-1,j}$ and the influence of the fault mode on the predicted action $\hat{a}_{i,j}$ while adding $\hat{a}_{i,j}$ and $\hat{S}_{i,j}$ to the fault mode's trajectory (Lines 6-11). Next, SFMa checks if $\hat{S}_{i,j}$ and S_i are equal, and if they are not, SFMa removes $\langle f_j, \mathcal{T}_{f_j} \rangle$ from the candidate fault modes (Lines 12-13). Finally, in every step $1 < i \leq n$, SFMa checks the size of $\mathcal{G}$, and halts if the size becomes 1, finding the correct fault mode. In case SFMa did not halt, it means there is more than 1 candidate fault mode that is consistent with the observation.

▶ **Example 8.** Refer to Example 2. After the initialization, SFMa simulates the next states for every fault model. Since f_2 changes R actions, the trajectory T_{f_2} deviates from the normal execution as early as state $\hat{S}_2$, meaning that by the time SFMa reaches time step $i = 5$, the state $\hat{S}_5$ returned by simulation does not match the observed state S_5. Consequently, the tuple $\langle f_2, \mathcal{T}_{f_2} \rangle$ will be removed from $\mathcal{G}$. SFMa will then continue running until reaching time step 7, at which point SFMa will find that the state $\hat{S}_{7,3}$ that is part of $\mathcal{T}_{f_3}$ is not equal to S_7. This is because f_3 changes R actions differently than f_1. At this point, SFMa will remove $\langle f_3, \mathcal{T}_{f_3} \rangle$ from $\mathcal{G}$, and will return the last and only fault mode, f_1.

5 Theoretic Analysis

In the following two sections we provide theoretic evaluation, that includes runtime analysis and some correctness proofs.

Algorithm 2 SFMa.

Input: π - The policy.
Input: χ - The simulator.
Input: $\mathcal{O} = (S_0, ..., S_n)$ - The observed states.
Input: $\mathcal{F} = \{f_1, f_2, ...f_k\}$ - Candidate fault modes.
Result: $\mathcal{D} = \{d_1, d_2, ...\}$ - The set of diagnoses.

$\mathcal{G} \leftarrow \emptyset$
for $j = 1$ *to* k **do**
  $\mathcal{T}_{f_j} \leftarrow (S_0)$
  $\mathcal{G} \leftarrow \mathcal{G} \cup \langle f_j, \mathcal{T}_{f_j} \rangle$
for $i = 1$ *to* n **do**
  for $j = 1$ *to* $|\mathcal{G}|$ **do**
    $\hat{a}_i \leftarrow \pi(\hat{S}_{i-1,j})$
    $\hat{a}_{i,j} \leftarrow f_j(\hat{a}_i)$
    $\hat{S}_{i,j} \leftarrow \chi(\hat{S}_{i-1,j}, \hat{a}_{i,j})$
    insert $\hat{a}_{i,j}$ *to* $\mathcal{T}_{f_j}$
    insert $\hat{S}_{i,j}$ *to* $\mathcal{T}_{f_j}$
    if $S_i \in \mathcal{O} \wedge S_i \neq \hat{S}_{i,j}$ **then**
      $\mathcal{G} = \mathcal{G} \setminus \langle f_j, \mathcal{T}_{f_j} \rangle$
  if $|\mathcal{G}| = 1$ **then**
    $\mathcal{D} \leftarrow \{f_j \ s.t. \ \langle f_j, \mathcal{T}_{f_j} \rangle \in \mathcal{G}\}$
    return $\mathcal{D}$
$\mathcal{D} \leftarrow \{f_j \ s.t. \ \langle f_j, \mathcal{T}_{f_j} \rangle \in \mathcal{G}\}$
return $\mathcal{D}$

5.1 Runtime analysis

The components that influence the run-time are the length between the first and last observed states n, the number of candidate fault modes k (in the case of SFMa) and three domain-dependent components: action prediction, action simulation, and state comparison. We refer to those domain dependent components using the constant $R(domain)$. In the analysis we will focus on the runtime as a function of n and k.

WFMa (Algorithm 1) is straightforward - it goes over the number of states n, and compares each state until it finds two states that are not matching. This gives a total runtime of $O(n \cdot R(domain))$.

SFMa (Algorithm 2) goes over the states where for each state it executes the policy, and then goes over the fault modes, where it does a finite number of simulation and prediction actions. This gives a runtime of $O(n \cdot k \cdot R(domain))$.

5.2 Correctness Proofs

▶ **Theorem 9.** *The index i_e of the first failure in the observed execution holds $i < i_e \leq i'$ where i, i' are the first and last diagnoses returned by WFMa.*

Proof. Assume, by contradiction, that the index of the first failure in the observed execution i_e does not hold $i < i_e \leq i'$. This means that $i_e \leq i \vee i' < i_e$.

Case 1 ($i_e \leq i$). In this case the algorithm still keeps the index $l < i_e$ of the last matching state. The algorithm will reach a step i'_e that holds $i_e \leq i'_e \leq i$, for which an observation exists. The observation $S_{i'_e}$ will not match the simulated state $\hat{S}_{i'_e}$, and the algorithm will halt, returning $l, .., i'_e$ in contradiction to the returned diagnoses $i, .., i'$.

Case 2 ($i' < i_e$). In that case, all of the observable states until step i_e are matching the simulation. That includes state $S_{i'}$. Consequently, when the algorithm checks $S_{i'} = \hat{S}_{i'}$ in line 7, it will set the beginning index to be some $b > i$. As a result, the returned index in the first diagnosis will be b that holds $b \geq i' > i$, in contradiction to the returned indices $i, i+1, ..., i'$. ◀

▶ **Theorem 10.** *SFMa is sound.*

Proof. To prove this, we show that for every fault mode f_j returned by SFMa as diagnosis, it follows that $\forall S_i \in \mathcal{O},\ S_i = \hat{S}_i$, where $\hat{S}_i$ is the i-th state in the trajectory simulated by $\chi(S_0, \pi, n, f_j)$.

Let f_j be a fault mode returned by SFMa. Assume by contradiction that $\exists S_i \in \mathcal{O}$ such that $S_i \neq \hat{S}_i$. Because of that, SFMa will remove the tuple $\langle f_j, \mathcal{T}_{f_j} \rangle$ from $\mathcal{G}$ at the i-th iteration in line 13. Following that, f_j will not be returned as a diagnosis, in contradiction to our initial assumption. ◀

▶ **Theorem 11.** *SFMa is complete.*

Proof. To prove that we show that every fault mode f_j for which it holds that $\forall S_i \in \mathcal{O},\ S_i = \hat{S}_i$, where $\hat{S}_i$ is the i-th state in the trajectory simulated by $\chi(S_0, \pi, n, f_j)$, will be returned by SFMa.

Let f_j be a fault mode as such, and assume by contradiction that it was not returned by SFMa. This means that for some time-step i SFMa removed the tuple $\langle f_j, \mathcal{T}_{f_j} \rangle$ from $\mathcal{G}$. This means that in line 12 SFMa determined that $S_i \neq \hat{S}_i$, by contradiction to f_j being defined as above. ◀

6 Evaluation

In the next two sections we outline the experimental setup and show empirical results.

6.1 Domains

We experimented with the domains *Acrobot*, *CartPole*, *MountainCar* and *Taxi*, all of which can be found in the Gymnasium website[2]. Table 1 shows general statistics about each domain. The "State Vars" row in Table 1 displays the type of variables describing the state – continuous or discrete. The "Env. Type" row displays whether the environment in each domain is dynamic or static, where "dynamic" here means the state may change even if the agent does not do any action, e.g., due to external forces that operate over the agent. The "Avg. Traj." row displays the average length of the trajectories in this domain.

Table 1 General statistics of our benchmark domains.

Domain	Acrobot	CartPole	MountainCar	Taxi
# actions	3	2	3	6
State Vars	Continuous	Continuous	Continuous	Discrete
Env. Type	Dynamic	Dynamic	Dynamic	Static
Avg. Traj.	148	54	183	63

[2] https://gymnasium.farama.org/

6.2 Experimental Setup

Before generating the problems, we obtained a policy for each domain, either by training or by using pre-trained policies available online. The generation of the problems consisted of the following steps:

1. Initialize a set $\mathcal{F}$ of k fault modes.
2. Choose a fault mode $f_e \in \mathcal{F}$ to be used during policy execution.
3. Execute the policy with the chosen fault mode for up to 200 steps, 10 times. We deemed 200 steps a sufficient trajectory length for the purpose of our experiments, during which at least one fault will occur.
4. Set the percent of visible states v. In an execution with n states, the states that must be observable are states S_0 and S_n. However, execution lengths can vary since we do not control them. Because of that we experiment with the percent of observable states rather than the number. We set $v = 0\%$ for an execution where only S_0 and S_n are observable, and $v = 100\%$ for an execution where every state is observable. We then experiment with different values of v.

6.3 Parameters

For our experiments, we considered fault modes that change some of the actions to behave like other actions. We use this type of fault modes for convenience. For each domain except CartPole, we tested 10 different fault modes f_e. For CartPole, we could only test 3 fault modes due to the number of different actions being only two (moving to the left and moving to the right) and the fact that for the experiments we defined fault modes to be *Many-to-one* or *Onto* functions $f : \mathcal{A} \to \mathcal{A}$.

We ask the following questions:

Q1 How does the number of fault modes impact the performance of SFMa?
Q2 How does the number of visible states affect the performance of WFMa and SFMa?

To answer **Q1** we experimented with $k \in \{2, 4, 6, 8, 10\}$ for SFMa in domains *Acrobot*, *MountainCar* and *Taxi* and with $k \in \{2, 3\}$ for SFMa in the *CartPole* domain. To answer **Q2** we experimented with $v \in \{0, 5, 10, 15, 20, 30, 40, 60, 80, 100\}$ for both WFMa and SFMa. Lastly, we set fixed parameter of $k = 0$ for WFMa since WFM assumes no fault modes.

6.4 Results

In this section, we present the results that answer our research questions. The Y-axis shows the runtime, and the X-axis shows the different parameter values we experiment with. The X-axis parameters differ from figure to figure. In addition, we crop the values to show whole numbers only, since the runtime is shown in milliseconds.

Figure 2 shows the runtime of SFMa with the increase in the number of candidate fault modes k. We can observe a linear increase in the runtime with the increase in fault modes number. This is expected since an increase in fault modes number has a linear effect on Algorithm 2.

Figure 3 shows the runtime of WFMa with the increase of the percent of visible states v. We observe that the runtime increases significantly from 5% to 0% visibility. We explain this result as follows. At 5% visibility the visible states include states that are not the initial or the last. Because of that, WFMa has more chances to determine whether the agent experienced a fault earlier in the diagnosis process when it encounters a state that does not match its counterpart's simulated state. The one domain that stands out is *CartPole*, where there is no significant increase in runtime for any value of v and where the runtime is very

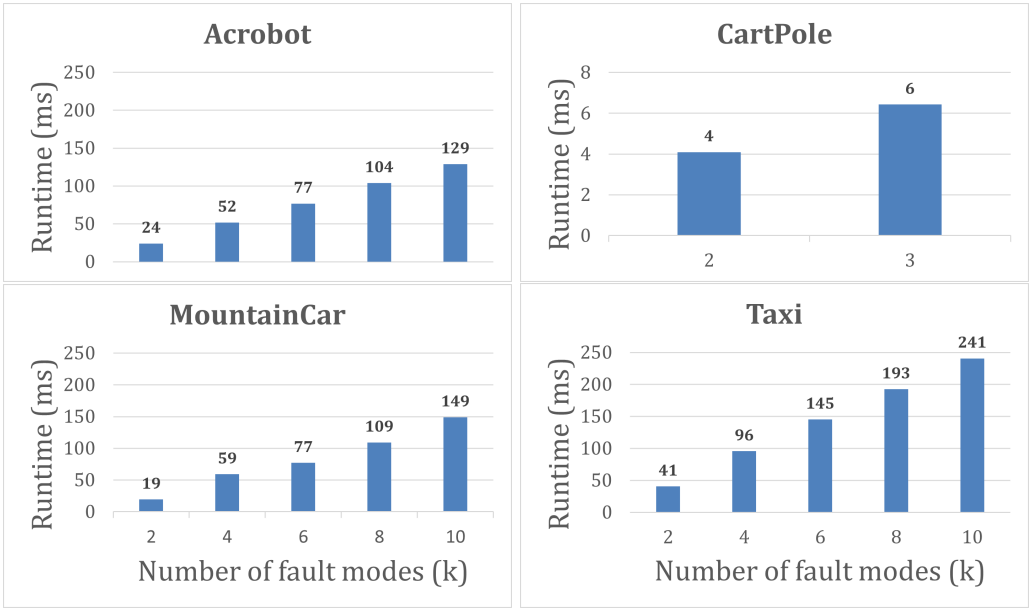


Figure 2 Runtime in milliseconds of SFMa with increase of k.

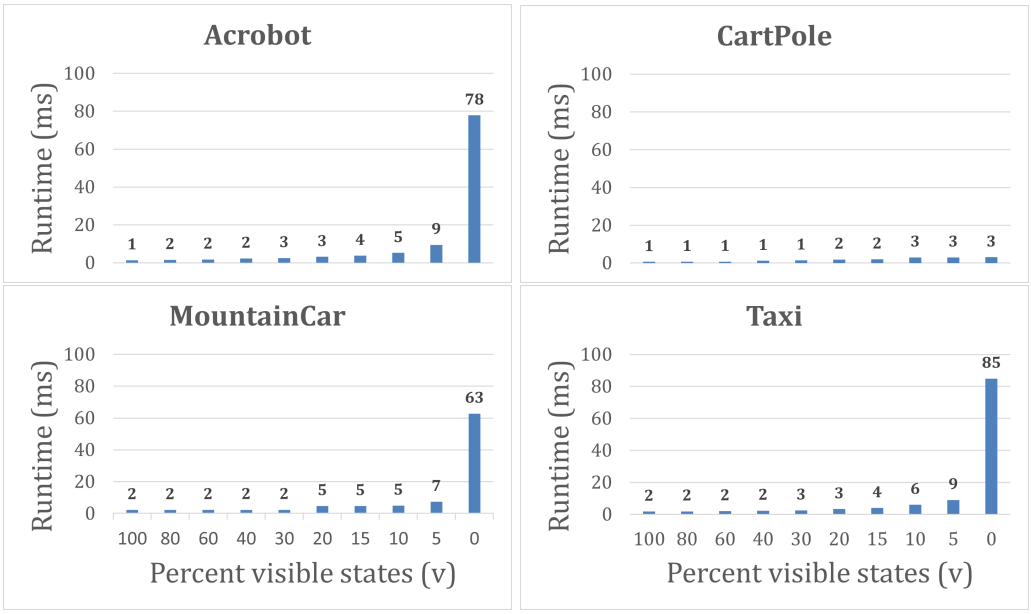


Figure 3 Runtime in milliseconds of WFMa with increase of v.

low. To explain this, we remind that the number of actions in this domain is 2. This means that only up to 3 fault modes could be tested (as explained in Section 6.3) and shown in Figure 2. For such a few actions and candidate fault modes, the runtime stays low and is not influenced by the different percentages of visible states.

Figure 4 shows the runtime of SFMa with the increase of the percent of visible states v. Here we also observe a significant increase between 5% and 0% visible states. The only exception is the *CartPole* domain, where the runtime increases steadily across all values of v. This increase, however, is not significant. We explain this case the same way we explained it in the previous paragraph for Figure 3.

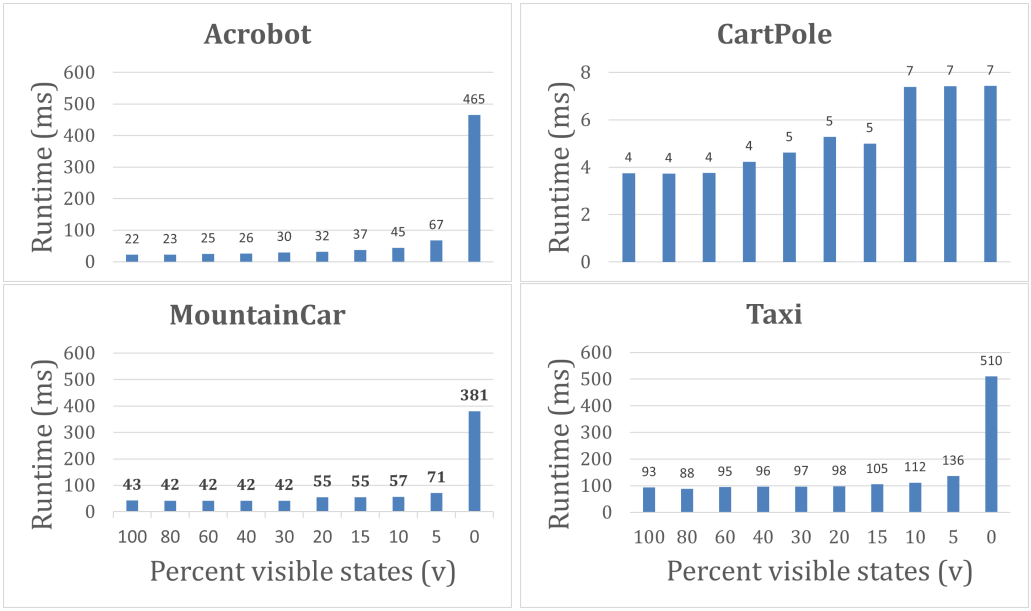


Figure 4 Runtime in milliseconds of SFMa with increase of v.

7 Discussion

In this section we discuss the strengths and weaknesses of the proposed algorithms, as well as some limitations.

The first thing to notice is the significant difference between weak and strong fault model assumptions. While WFMa is faster in terms of runtime, it outputs a less informative diagnosis than SFMa.

Another limiting assumption is the deterministic nature of the policy we use. This allows the reconstruction of a trajectory. For WFMa, it means that given a state S it can reconstruct the healthy execution. For SFMa, it means that given a state S and a fault mode f it can reconstruct an execution where the faulty actions fail according to f. This assumption is important for our algorithms. However, it limits them to cases where the policy is deterministic.

Last but not least, we assume non-intermittent faults. This significantly effects the runtime of SFMa since it means there is only one fauty trajectory, in which every action that should fail according to the fault mode, fails. Although the assumption of non-intermittent faults is a limiting assumption, cases where such faults occur can be thought of. For example, a flying device where one of its engines short-circuited will always have that engine not working, when the commands for that engine are sent.

8 Conclusions and Future Work

In this work, we addressed the problem of diagnosing faulty executions of policies generated by reinforcement learning algorithms. In particular, we focused on non-intermittent faults. We presented two algorithms that address this problem. The first, called WFMa, addresses the problem while assuming *weak fault model*. In this setting a diagnosis is a time-step where a faulty action failed for the first time. The second algorithm, called SFMa, addresses the

problem while assuming *strong fault model*. In this setting, a diagnosis is a fault mode that can explain the observation. We tested the algorithms on four well known Reinforcement Learning domains from the Gym benchmarks.

We found out that the runtime of both algorithms increases with the decrease in percentage of observed states. In particular for both algorithms and for every domain except *CartPole*, the runtime increase is the highest between 5% and 0% observability. Apart from that, we found that the runtime of SFMa increases linearly with the increase of the number of candidate fault modes.

As future work, we plan to address intermittent faults setting, and particularly for strong fault model. In this setting, the complexity of the problem grows by a factor of $O(2^n)$. That is because, in the worst case, for a fault mode that affects every one of the n actions, and where only S_0 and S_n are observed, there is a need to address in every time step the possibility of a faulty and a healthy execution of the faulty actions. Apart from that, we plan to experiment with the algorithms on other domains from the Gymnasium benchmark domains and, possibly, test our algorithms on real-world problems. Lastly, we plan to extend this research direction by introducing problems involving multiple agents.

References

1 Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, pages 89–98. IEEE, 2007.

2 Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. Distributed diagnosis of coupled mobile robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3787–3794. IEEE, 2006. doi:10.1109/ROBOT.2006.1642281.

3 Johan De Kleer, Alan K Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial intelligence*, 56(2-3):197–222, 1992. doi:10.1016/0004-3702(92)90027-U.

4 Orel Elimelech, Roni Stern, Meir Kalech, and Yedidya Bar-Zeev. Diagnosing resource usage failures in multi-agent systems. *Expert Systems with Applications*, 77:44–56, 2017. doi:10.1016/J.ESWA.2017.01.047.

5 Meir Kalech and Gal A Kaminka. Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, 27(3):393–421, 2011. doi:10.1111/J.1467-8640.2011.00386.X.

6 Meir Kalech and Avraham Natan. Model-based diagnosis of multi-agent systems: A survey. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12334–12341, 2022. doi:10.1609/AAAI.V36I11.21498.

7 Roberto Micalizio and Pietro Torasso. Cooperative monitoring to diagnose multiagent plans. *Journal of Artificial Intelligence Research*, 51:1–70, 2014. doi:10.1613/JAIR.4339.

8 Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

9 Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. doi:10.1038/NATURE14236.

10 Avraham Natan and Meir Kalech. Privacy-aware distributed diagnosis of multi-agent plans. *Expert Systems with Applications*, 192:116313, 2022. doi:10.1016/J.ESWA.2021.116313.

11 Avraham Natan, Meir Kalech, and Roman Barták. Diagnosis of intermittent faults in multi-agent systems: An sfl approach. *Artificial Intelligence*, 324:103994, 2023. doi:10.1016/J.ARTINT.2023.103994.

12 Avraham Natan, Roni Stern, and Meir Kalech. Distributed spectrum-based fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6491–6498, 2023. doi:10.1609/AAAI.V37I5.25798.

13 Alexandre Perez, Rui Abreu, and Marcelo d'Amorim. Prevalence of single-fault fixes and its impact on fault localization. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 12–22, 2017. doi:10.1109/ICST.2017.9.

14 Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

15 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

16 Patrick Rodler. How should i compute my candidates? a taxonomy and classification of diagnosis computation algorithms. *arXiv preprint arXiv:2207.12583*, 2022. doi:10.48550/arXiv.2207.12583.

17 Nico Roos, Annette Ten Teije, and Cees Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 655–661, 2003. doi:10.1145/860575.860681.

18 Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19:30–52, 2009. doi:10.1007/S10458-007-9017-6.

19 Michael Schmid, Emanuel Gebauer, Christian Hanzl, and Christian Endisch. Active model-based fault diagnosis in reconfigurable battery systems. *IEEE Transactions on Power Electronics*, 36(3):2584–2597, 2020.

20 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. arXiv:1707.06347.

21 Peter Struss and Oskar Dressler. " physical negation" integrating fault models into the general diagnostic engine. In *IJCAI*, volume 89, pages 1318–1323, 1989. URL: http://ijcai.org/Proceedings/89-2/Papers/075.pdf.

22 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

23 Gianluca Torta and Roberto Micalizio. Smt-based diagnosis of multi-agent temporal plans. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2097–2099, 2018. URL: http://dl.acm.org/citation.cfm?id=3238084.

24 Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. doi:10.5281/zenodo.8127026.

Faster Diagnosis with Answer Set Programming

Liliana Marie Prikler
Institute of Software Technology, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Technology, Graz University of Technology, Austria

Abstract

From hardware to software to human patients, diagnosis has been one of the first areas of interest in artificial intelligence, and has remained a relevant topic since. Recent research in model-based diagnosis has shown that answer set programming not only allows for an easy expression of diagnosis problems, but also efficient solving. In this paper, we improve on previous results by making use of various modern answer set programming techniques. Our experiments compare multi-shot solving, heuristics and preferences, with results indicating that heuristics provide the fastest solutions on most instances we studied.

2012 ACM Subject Classification Computing methodologies → Logic programming and answer set programming; Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases Answer set programming, model-based diagnosis, performance comparison

Digital Object Identifier 10.4230/OASIcs.DX.2024.24

Category Short Paper

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.13850098` [20]

Funding This paper is part of the AI4CSM project that has received funding within the ECSEL JU in collaboration with the European Union's H2020 Framework Programme (H2020/2014-2020) and National Authorities, under grant agreement No.101007326. The work was partially funded by the Austrian Federal Ministry of Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) under the program "ICT of the Future" project 877587.

1 Introduction

With the advent of new and potentially dangerous technologies such as self-driving vehicles, the time it takes to diagnose a fault is becoming increasingly relevant for systems aimed at maintaining safety and security. For example, an autonomous vehicle that drives on a highway needs to compensate failures until a safe state can be reached. For such compensating actions, diagnosis capabilities are required that are fast enough for providing relevant information, e.g., the severity of a failure and its origin. In model-based diagnosis, where we rely on knowledge representation and reasoning, we need both faster solvers and faster diagnosis algorithms to keep up with these challenges.

Previous investigations on diagnosis via SAT solvers showed that direct encodings of the diagnosis problem can outperform diagnosis algorithms [16]. Similarly, diagnosis via answer set programming is able keep up with or even outperform hitting-set based algorithms [24]. Most recently, Bayerkuhnlein and Wolter [1] introduced other codings and improvements of diagnosis utilizing answer set programming.

It is also worth noting that diagnosis has been a benchmark for solver optimization. Techniques such as heuristics [10] and preferences [2] have been studied with diagnosis as a potential application.

In this paper, we take a combined approach and look at both algorithms and solver-based approaches for efficient diagnosis. Our contributions are two-fold:


© Liliana Marie Prikler and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



- we improve the performance of the IDIAG [17, 24] algorithm by providing an incremental implementation that makes use of multi-shot solving [9], and
- we compare IDIAG to solver optimizations [10, 2], thereby bridging a gap in research which has previously focused on solvers **or** algorithms, but not both at the same time.

The rest of this paper is structured as follows. In Section 2, we introduce the basic concepts of model-based diagnosis and answer set programming, as well as providing a short overview of alternative approaches towards model-based diagnosis through answer set programming. In Section 3, we provide an improved formulation of the prover in [24] that makes use of Clingo's multi-shot solving. In Section 4, we report the results of our comparison. Finally, we conclude the paper in Section 5.

2 Background

In this section, we lay out the basics for understanding the rest of the paper. We start with a description of model-based reasoning and the IDIAG algorithm. Then we describe answer set programming, followed by declarative approaches for obtaining optimal answer sets.

2.1 Model-based reasoning

The underlying idea of model-based reasoning [7, 21] is to solve problems by directly applying available knowledge. One starts with a description of a system from first principles, using some available (typically logical) formalism to formulate this description.

▶ **Definition 1** (Diagnosis system). *A tuple (D, C) is a diagnosis system, where D is the (logic) description of a system, and C is a set of components.*

Here, D describes the nominal behaviour of the system, assuming that all components function as expected. To make these assumptions explicit, predicates $\mathrm{ab}(c)$ and $\neg\mathrm{ab}(c)$ with a given component $c \in C$ are used to indicate that c is behaving "abnormally" (i.e. in a faulty manner) or "not abnormally" (i.e. nominally or healthy).

With this description, we start observing the system under diagnosis. We assume that several observations can be gathered from said system at a time, without impacting its normal behavior.[1] As soon as we find a discrepancy between the expected and actual behaviour of a system, we find ourselves with a diagnosis problem.

▶ **Definition 2** (Diagnosis problem). *Let (D, C) be a diagnosis system and O a set of observations. The tuple (D, C, O) is a diagnosis problem.*

Note, that this definition does not assume that our observations are inconsistent in combination with the system description. Indeed, using boolean satisfiability or answer set programming to model D, we would have to solve an NP-hard subproblem to decide, whether our observations are consistent with reasonable expectations.

Given a diagnosis problem, we want to find out which components of the system – if any – are faulty. This leads us to the actual *diagnoses* of model-based diagnosis.

▶ **Definition 3** (Diagnoses). *Let (D, C, O) be a diagnosis problem. A subset $\Delta \subseteq C$ is a diagnosis if and only if $D \cup O \cup \{ab(c)|c \in \Delta\} \cup \{\neg ab(c)|c \notin \Delta\}$ is consistent. A diagnosis is (subset-)minimal, or parsimonious, if and only if there exists no $\Delta' \subset \Delta$ that is itself a diagnosis. A diagnosis is superfluous if it is not parsimonious.*[2]

[1] This assumption may be invalid under quantum mechanics, but is assumed to be true for boolean circuits or human patients.

[2] In some literature, superfluous diagnoses may not be considered diagnoses at all.

Several algorithms exist to solve diagnosis problems [21, 12, 23, 19], but prior research suggests that direct computation of diagnoses is often just as viable. [16, 17, 24] In this paper, we focus on model-based diagnosis via answer set programming [24], which uses the procedure IDIAG and an ASP-specific prover that we discuss in the next section. Conceptually, IDIAG and Reiter's algorithm [21] are quite similar: as the number of iterations increases, so does the size of yielded diagnoses.[3]

Procedure IDIAG(D, C, O, n).

Data: a system description D, a set of components C, a set of observations O, the desired cardinality n
Result: all subset-minimal diagnoses of size up to n
Let $\Delta = \varnothing, P = D \cup O$;
for $i = 0$ **to** n **do**
  Let $\Delta_i :=$ prover(C, P, i) ;
  if $i = 0 \wedge \Delta_i = \{\varnothing\}$ **then**
    return Δ_i ;
  end
  Set $\Delta = \Delta \cup \Delta_i$;
  Set $P = P \cup \{\mathsf{ab}(c_1) \wedge \cdots \wedge \mathsf{ab}(c_i) \rightarrow \bot | \{c_1 \dots c_i\} \in \Delta_i\}$;
end
return Δ ;

2.2 Answer set programming

Answer set programming [4, 14] is a declarative problem-solving technique based on non-monotonic reasoning. Formally, we distinguish grounded answer set programs and answer set programs as they are written in practice, with predicates, variables, and other extensions.

A grounded answer set program consists of rules of the shape

$$h \leftarrow b_1 \wedge \cdots \wedge b_n \wedge \textbf{not } c_1 \wedge \cdots \wedge \textbf{not } c_m,$$

with h, b_i, c_j for $1 \leq i \leq n, 1 \leq j \leq m$ being drawn from some set of atoms $\mathcal{A}$. These rules look quite similar to Horn clauses – in fact, for $m = 0$, they are Horn clauses.

▶ **Definition 4** (Gelfond-Lifschitz transformation [11]). *Let P be a grounded answer set program and τ a set of atoms. Construct P^τ from P as follows:*

- *if a rule contains* ***not*** *x, with $x \notin \tau$, remove* ***not*** *x from that rule*
- *if a rule contains* ***not*** *x, with $x \in \tau$, remove that rule from P^τ*

By construction, the Gelfond-Lifschitz transformation arrives at a positive logic program, that is, a logic program consisting only of Horn clauses.

▶ **Definition 5** (Stable model). *Let P be a grounded answer set program and τ a set of atoms. Let P^τ be the Gelfond-Lifschitz transformation of P using τ. τ is a stable model of P if and only if τ is the smallest model of P^τ under classical (or Horn) satisfiability.*

[3] In an interesting twist, however, only IDIAG is able to yield these partial results. With an algorithm based on conflict sets, such as Reiter's algorithm, future iterations may invalidate prior results, whereas in a direct computation, minimality can be guaranteed by construction.

To account for variables, we make use of the identities

$$\exists x.\varphi \Leftrightarrow \varphi_{x=x_1} \vee \cdots \vee \varphi_{x=x_n}$$

and

$$\forall x.\varphi \Leftrightarrow \varphi_{x=x_1} \wedge \cdots \wedge \varphi_{x=x_n},$$

where x_i for $1 \leq i \leq n$ denotes all possible values the variable x may take [13].

▶ **Definition 6** (Herbrand universes and bases). *Let P be a logic program. The Herbrand universe of P, denoted $\mathcal{U}(P)$ is the set of all terms which can be formed from constants and function symbols in P.*

The Herbrand base $\mathcal{B}(P)$ is the set of all variable-free atoms, which can be constructed from predicates in P and terms in $\mathcal{U}(P)$.

By definition, the Herbrand universe holds every value that is of potential meaning for the program P. For a given rule r containing variables, we may assume that the variables in the head are implicitly qualified using $\forall$, whereas the variables appearing only in the body are implicitly qualified using $\exists$.

▶ **Definition 7** (Grounding). *A ground instance of a rule $r \in P$ is obtained by applying a substitution of the variables in r with atoms taken from $\mathcal{U}(P)$. Let $\mathcal{G}(r)$ be the set of all possible ground instances of r. Then $\mathcal{G}(P) = \bigcup_{r \in P} \mathcal{G}(r)$.*

As each variable is expanded separately from each other, we arrive at a program where $r \Leftrightarrow \mathcal{G}(r)$. Taking $\mathcal{A} = \mathcal{U}(P)$ and $\tau \subset \mathcal{B}(P)$, the stable models of an answer set program P thus become the stable Herbrand models of its grounded program $\mathcal{G}(P)$.

This technique is easy enough to understand for illustrative purposes, but in practice wasteful. Modern solvers make use of more efficient techniques such as domain-restricted grounding [18] and also allow other syntactic constructs (cf. [6, 8]). Similar to variables, these extensions are grounded before the program is solved, leading to no change in semantics for the grounded program, while aiding the programmer in formulating their program.

Before we discuss extensions for optimization, we first quickly show aggregates and cardinality constraints, which necessarily appear in diagnosis. The ASP-Core-2 standard [6], which provides a common syntax for solvers, allows for disjunctive heads of the shape

$$h_1 \vee \cdots \vee h_k$$

in which exactly one $h_i, 1 \leq i \leq k$ becomes true as a result of applying the rule, or more generally choice rules of the shape

$$\alpha_1 \prec_1 \{h_1; \ldots; h_k\} \prec_2 \alpha_2.$$

Here, α_1 and α_2 can impose a cardinality constraint on the number of h_i that hold, with $\prec_{1,2} \in \{=, \neq, <, >, \geq, \leq\}$. Either side can also be omitted to only have a single cardinality constraint. In the body, such constraints can similarly be imposed by writing

$$\alpha_1 \prec_1 \texttt{\#count}\{l_1; \ldots; l_k\} \prec_2 \alpha_2.$$

These numerical comparisons retain their intuitive mathematical meaning as far as the answer set solver allows. For the practical purposes of this paper, it suffices that operations on integers in the range $[-2^{31}, 2^{31} - 1]$ are supported.

2.3 Preferred and optimal solutions

Similar to how diagnosis algorithms are tasked to find parsimonious diagnoses, we find optimization problems in other domains where logic solvers are used. To cope with such problems, modern solvers implement methods to describe them and obtain optimal solutions. Within the Potassco solver family – to which Clingo [9], the solver we will use for our implementation, belongs – two techniques stand out as applicable to diagnosis problems: domain-specific heuristics [10] and preferences [2].

With domain-specific heuristics, users guide the internal decision making of their solver. Solvers use *conflict-driven clause learning* [15], a process that alternates between guessing the truth value of a particular literal and then propagating the effects of that guess, so that eventually either a satisfying assignment (or in the context of ASP a stable model) or a conflict is reached – this conflict is then used to derive a clause which is added to the input program. The solver then backtracks to an earlier decision and starts anew. Heuristics can guide this process in two manners: first by choosing which literals to assign, second by choosing the values to assign.

With preferences, users turn a decision problem "is there an answer set?" to an optimization problem "is there an answer set, preferably a small one?". These preferences are themselves written as a program which is evaluated on (reified) answer sets to determine which of them (if any) is preferred over the other. Several such programs are already implemented in asprin [3, 2], so that one may also use pre-defined preferences, as shown in Listing 1.

Listing 1 Stating the preference for parsimonious diagnoses.

```
#preference(p,subset) { ab(X) }.
#optimize(p).
```

Some of these preferences could be implemented in Clingo itself as weak constraints (e.g. cardinality-minimal diagnoses), but notably, parsimonious diagnoses are not one of them.

3 An incremental prover

In this section we describe our incremental implementation, which is based on the multi-shot solving of Clingo [9]. Similar to modern SAT solvers, which support incremental solving, multi-shot solving allows incremental grounding and solving of answer set programs. This opens up a tremendous opportunity for IDIAG, which for the most part only adds constraints to the underlying program.

In Listing 2, we show the necessary program parts for an incremental prover. As part of the base program, we assume that `comp/1` defines the components of the system and allow the solver to guess whether a component works as expected (in which case `nab(C)` holds), or does not work as expected (in which case `ab(C)` holds) with a choice rule. As a diagnosis consists of only the broken components, we show `ab/1`. Further, we heuristically choose `ab(C)` not to hold. This heuristic is in fact not relevant to IDIAG itself, as it needs to be explicitly enabled, but it is the main piece we will compare IDIAG against.

The *incremental* part comes with the program `idiag_step`. This program is grounded with a single argument n and corresponds to parameterization of `$n$` in [24].[4] It provides both an external literal and a constraint that becomes enabled with this external literal,

[4] Note that we use `_idiag_step_`, as it is a valid symbol that is unlikely to be bound to a constant. In clingo, whole-program constants shadow the constants given in a program parameter.

disallowing solutions of cardinality higher than the current target cardinality. The inequality is only formulated in one direction, as there ought to be no stable model in the other direction (with smaller cardinality) following the updates from IDIAG.

Listing 2 Supporting logic program for IDIAG using multi-shot solving. A choice rule is used to enforce that $ab(c) \wedge \neg ab(c)$ holds – Clingo implicitly uses $\leq$ for comparing two values.

```
#program base.
#defined comp/1.
1 { ab(C) ; nab(C) } 1 :- comp(C).
#show ab/1.
#heuristic ab(C). [1, false]

#program idiag_step(_idiag_step_).
#external max_ab(_idiag_step_).
:- max_ab(_idiag_step_), #count {C : ab(C)} > _idiag_step_.
```

With these program parts, we can now discuss the procedure prover. In collaboration with IDIAG, it maintains a representation of the diagnosis problem and the diagnoses, which were already found and ought to be ignored. The literal `max_ab(`n`)` becomes active for at most one iteration – at most one, because it might not exist.[5]

Procedure prover(C, M, n).

Data: a set of components C, a model M, a cardinality n
Invariant: no diagnosis of cardinality $n-1$ exists (maintained by `IDIAG`)
Result: all diagnoses of cardinality n
Let M' be the `clingo` representation of M ;
Ground idiag_step (n) ;
if *there exists an external literal* `max_ab(`n`)` **then**
  Assign $\top$ to `max_ab`(n) ;
  Let Σ be the answer sets of M' ;
  Let $\Delta = \{\{c_1, \ldots, c_n\} | \{\mathsf{ab}(c_1), \ldots, \mathsf{ab}(c_n)\} \in \Sigma\}$;
  `/* same as max_ab(`n`) := ⊥ for the remainder of the program */`
  Release `max_ab`(n) ;
  return Δ ;
end
else
  return $\varnothing$;

4 Experimental results

We implemented IDIAG and prover using the Clingo C API, inlining the prover into IDIAG, so that a situation in which no more diagnoses exist can be detected early (cf. [20] for the implementation and the scripts used for benchmarking). Our implementation is loosely

[5] Indeed, Clingo is wise enough to detect when it has already found all minimal diagnoses – while this is not obvious from the Python API, it is an error we need to deal with in the C API.

coupled with Listing 2 – it assumes, that an `idiag_step` program exists, which defines an external literal `max_ab`, and that the shown symbols of a model correspond to the diagnosis to report.

We compare our implementation with other techniques using the well-studied ISCAS'85 benchmark circuits [5]. In particular, we reuse the test data from [24], but rewrite it to conform stronger to the structure actually laid out in that paper.[6] For our implementation to work as intended, we need `comp(C)` to actually be established as a fact (for Listing 2), and we need `ab/1` to be the only shown predicate (as per the limitation mentioned above).

As an example, Listing 3 shows a test input before rewriting, and Listing 4 after rewriting. During this rewrite we also drop the symbol `no_ab(N)`, used to indicate the cardinality of a diagnosis. Where needed, we reintroduce this literal rather than using the `idiag_step` program, which is only handled by our implementation. Unless otherwise noted, we also provide a constraint on the maximum cardinality of *any* diagnosis, via the standard input. Regardless of how we inform the implementation of the maximum cardinality, we fix it to the number 3.

Listing 3 Example input before rewriting.

```
:- lv_1gat_0, lv_1gat_1.
:- lv_4gat_0, lv_4gat_1.
% ...

nab_118gat :- not ab_118gat.
ab_118gat :- not nab_118gat.
% ...

lv_118gat_0 :- nab_118gat, lv_1gat_1.
lv_118gat_1 :- nab_118gat, lv_1gat_0.
lv_1gat_0 :- nab_118gat, lv_118gat_1.
lv_1gat_1 :- nab_118gat, lv_118gat_0.
% ...

lv_1gat_1.
% ...
```

We run our benchmarks on a Lenovo ThinkPad T450 with an Intel Core i5-5200U processor, 8 GB of RAM and 2 GB of Swap space. While the system runs Ubuntu, we use Python 3.10.7, Clingo 5.7.1, and asprin 3.1.1-1.bc5a0cf (corresponding to the newest commit at the time of writing), as packaged in GNU Guix on commit `6d81f2a...` (see Listing 5 for the complete channel list).

We first compare our implementation to a slightly modified version of the original Python implementation [24]. To make the comparison between the two tools fair, we use Clingo's statistics in both, rather than timing the entire process, which would also take into account file I/O and grounding times, further penalizing the reference implementation. The results can be seen in Table 1 and appear to strongly favour our implementation in all but two benchmarks.

Next, we compare our tool against the approaches laid out in Section 2.3, i.e., heuristics (built into Clingo) and preferences (via asprin). It should be noted, that both solver approaches do not guarantee that diagnoses are sorted by their cardinality. Thus, when there

[6] The original tool and benchmarks can be found at `https://github.com/QAMCAS/ASP-Diagnose-Tool`.

Listing 4 Example input after rewriting.

```
:- val(V, X), val(V, Y), X < Y.

comp(gat118).
% ...

val(gat118, 0) :- nab(gat118), val(gat1, 1).
val(gat118, 1) :- nab(gat118), val(gat1, 0).
val(gat1, 0) :- nab(gat118), val(gat118, 1).
val(gat1, 1) :- nab(gat118), val(gat118, 0).
% ...

val(gat1, 1).
% ...
```

Listing 5 Guix channels used to obtain Clingo, Python, asprin, as well as the dependencies of our IDIAG implementation.

```
(list
 (channel
  (name 'guix)
  (url "https://git.savannah.gnu.org/git/guix.git")
  (branch "master")
  (commit
   "6d81f2a4ade07158fbadd560722cc386007caf68")
  (introduction
   (make-channel-introduction
    "9edb3f66fd807b096b48283debdcddccfea34bad"
    (openpgp-fingerprint
     "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA")))))
```

are more than the desired number of diagnoses, they may report more diagnoses of a higher cardinality – or lower quality if cardinality is a concern. We argue, however, that parsimony along with a stated maximum cardinality are likely enough quality criteria, as diagnoses need to be verified before they can be acted upon.[7]

Tables 2 and 3 show the results of our comparison with Clingo and asprin. With the exception of two outliers, Clingo is faster than both asprin and IDIAG, both on average and in the worst case. Each of asprin and IDIAG is the fastest tool once, with IDIAG on average being closer in performance to Clingo, and asprin being the slowest on average. The effects that make asprin faster than IDIAG appear to be more pronounced when looking at worst-case performance.

Finally, we look a little more closely at asprin, which also has built-in support for heuristics. Asprin provides multiple means of turning its preferences into heuristics (cf. [3, 2]). Our own results are shown in Tables 4 and 5. Interestingly, it appears as though heuristics only have a weak positive effect (lower runtime; better performance) in some instances, and a

[7] Assuming one has the means of obtaining more observations interactively and for each component individually, diagnoses of higher cardinality may even be beneficial, as they provide multiple components to investigate in parallel. One may also discover that a particular component is a likely cause of failure by counting the number of diagnoses it appears in, cf. [22].

Table 1 Average runtimes (in seconds) of the original IDIAG implementation [24] and our tool when tasked to compute subset-minimal diagnoses for the ISCAS85 benchmark circuits. The better result is highlighted using a bold font.

	reference[1,a,l)]			ours[2)]		
circuit	1	2	3	1	2	3
c432	0.5292	0.5452	0.5244	**0.1161**	**0.1184**	**0.1291**
c499	0.8898	0.8310	0.6785	**0.3136**	**0.3274**	**0.2077**
c880	1.3896	1.9994	1.9312	**0.2262**	**0.3866**	**0.3307**
c1355	**4.4549**	**4.0057**	4.2121	4.8465	14.9191	**2.0967**
c1908	11.0776	9.4301	21.0627	**3.1129**	**1.3992**	**5.6861**
c2670	10.7953	17.2354	28.2214	**0.8324**	**1.9525**	**3.5458**
c3540	37.1014	38.0905	101.4154	**4.9932**	**6.0543**	**27.2434**
c5315	116.5337	97.6262	266.2112	**4.2492**	**3.091**	**14.3082**
c6288	36.3123	63.5793	115.7475	**3.5328**	**12.5654**	**11.2871**
c7552	528.0185	759.5175	5159.9682	**11.0034**	**30.3936**	**81.2927**

1) `python idiag-ref.py --support=idiag-ref.lp --faultsize 3 --answersets 100 "$test"`
2) `idiag 100 --stats=2 idiag.lp "$test" -`
a) Maximum cardinality of diagnosis provided via `--faultsize` argument.
l) `no_ab` defined in `idiag-ref.lp`.

Table 2 Average runtimes of various tools when tasked to compute subset-minimal diagnoses for the ISCAS85 benchmark circuits. The best result is highlighted with a bold font. An italic font is used to highlight, that asprin is second in terms of performance rather than third.

	asprin[1,u,N)]			clingo[2,u)]			IDIAG[3,p)]		
circuit	1	2	3	1	2	3	1	2	3
c432	3.0025	2.6269	2.4364	**0.086**	**0.0929**	**0.0964**	0.1161	0.1184	0.1291
c499	3.8399	2.9064	1.4139	**0.2319**	0.4069	**0.1866**	0.3136	**0.3274**	0.2077
c880	4.1091	6.2516	5.7019	**0.1532**	**0.1899**	**0.1544**	0.2262	0.3866	0.3307
c1355	9.0238	**8.2308**	7.4812	**1.9398**	37.8516	**1.2266**	4.8465	14.9191	2.0967
c1908	13.7109	15.781	15.7449	**0.5426**	**0.5067**	**1.0541**	3.1129	1.3992	5.6861
c2670	12.6344	17.4196	17.9126	**0.5566**	**0.5437**	**0.5851**	0.8324	1.9525	3.5458
c3540	21.9377	28.6404	34.1104	**0.9637**	**0.9309**	**5.7958**	4.9932	6.0543	27.2434
c5315	30.5597	21.8695	43.8268	**1.5116**	**1.4836**	**2.0253**	4.2492	3.091	14.3082
c6288	35.8091	39.6309	39.4172	**1.9525**	**1.6256**	**1.5504**	3.5328	12.5654	11.2871
c7552	67.5605	74.2026	*73.0008*	**2.6636**	**2.4526**	**3.1381**	11.0034	30.3936	81.2927

1) `asprin 100 --stats idiag.lp preference.lp "$test" -`
2) `clingo 100 --stats=2 --enum-mode=domRec --heuristics=domain idiag.lp "$test" -`
3) `idiag 100 --stats=2 idiag.lp "$test" -`
p) Reports diagnoses in a partially-ordered manner (by cardinality).
u) Reports diagnoses in a seemingly unordered manner.
N) May report more than the desired number of diagnosis (due to non-optimal solutions)

Table 3 Maximal runtimes of various tools when tasked to compute subset-minimal diagnoses for the ISCAS85 benchmark circuits. The best result is highlighted with a bold font. An italic font is used to highlight, that asprin is second in terms of performance rather than third.

	asprin[1,u,N]			clingo[2,u]			IDIAG[3,p]		
circuit	1	2	3	1	2	3	1	2	3
c432	5.426	3.927	3.746	**0.099**	**0.124**	**0.113**	0.136	0.153	0.195
c499	4.595	4.278	4.129	**0.481**	1.041	**0.284**	0.858	**0.965**	0.302
c880	8.099	7.794	6.751	**0.180**	**0.402**	**0.168**	0.434	1.582	1.111
c1355	15.886	**9.646**	8.969	**15.382**	232.803	**4.253**	18.113	82.934	5.994
c1908	22.879	18.766	19.498	**1.272**	**0.619**	**3.048**	12.844	3.724	16.156
c2670	29.583	22.347	27.956	**0.715**	**0.720**	**0.785**	2.161	5.625	16.682
c3540	33.875	34.000	*60.765*	**1.756**	**1.082**	**45.566**	21.039	18.122	107.792
c5315	53.090	39.037	78.912	**2.502**	**3.582**	**5.012**	15.348	5.075	72.425
c6288	40.874	42.954	42.745	**2.160**	**1.920**	**1.679**	7.663	35.284	26.621
c7552	73.987	*90.521*	*115.544*	**3.754**	**4.630**	**9.238**	36.467	136.745	242.663

1) `asprin 100 --stats idiag.lp preference.lp "$test" -`
2) `clingo 100 --stats=2 --enum-mode=domRec --heuristics=domain idiag.lp "$test" -`
3) `idiag 100 --stats=2 idiag.lp "$test" -`
p) Reports diagnoses in a partially-ordered manner (by cardinality).
u) Reports diagnoses in a seemingly unordered manner.
N) May report more than the desired number of diagnosis (due to non-optimal solutions)

significantly negative effect (much larger runtime; much worse performance) in others. These findings contrast prior research suggesting that heuristics to have a pronounced positive effect – we were not able to discover such an effect in our data.

Combining our comparison results, it appears as though structural properties in some test cases favour one family implementations over others. In particular, the c1355 benchmarks show good performance in asprin and the reference implementation, both of which use multiple solver calls, whereas most other benchmarks are faster using Clingo (single-shot) or our multi-shot IDIAG, which keep intermediate information.

Closer inspection of our test data reveals three test cases

- `c1355_tc_1_33`,
- `c1355_tc_2_33`, and
- `c1355_tc_2_34`,

which do not have a diagnosis of cardinality 3, but do have a diagnosis of higher cardinality. We imagine that our test results would favour asprin and the reference implementation more strongly if more inputs shared this behavioural pattern. We leave the investigation as to why this pattern causes such huge influence on some configurations, but not others, to future research.

5 Conclusion

We have implemented an incremental prover for the previously described IDIAG algorithm, based on the Clingo answer set solver. Our experiments indicate, that this incremental prover is significantly faster than the reference, but also, that Clingo itself is faster than both of them at determining parsimonious diagnoses.

We have further identified a problematic pattern, in which the performance of Clingo and our IDIAG implementation is worse than that of state-of-the-art alternatives. Future work may investigate these patterns more closely and find more efficient ways of handling them.

Table 4 Average runtimes of asprin with various heuristics. The best result is highlighted with a bold font.

	asprin[1)]			asprin$_s$[2,n)]			asprin$_l$[3,n)]		
circuit	1	2	3	1	2	3	1	2	3
c432	3.0025	2.6269	2.4364	2.8272	2.7462	2.5235	**2.1994**	**2.5593**	**2.4048**
c499	3.8399	**2.9064**	**1.4139**	3.7184	2.9622	1.429	**3.7165**	2.9736	1.4403
c880	4.1091	6.2516	**5.7019**	4.050	6.2695	5.7221	**3.9736**	**6.110**	5.740
c1355	**9.0238**	**8.2308**	**7.4812**	9.1101	8.3522	8.4739	10.1812	18.2236	9.5113
c1908	13.7109	**15.781**	**15.7449**	15.3369	16.5029	16.6365	**11.6994**	17.0772	22.685
c2670	12.6344	17.4196	17.9126	12.8349	18.9981	18.1386	**9.637**	**16.6658**	**17.0139**
c3540	**21.9377**	**28.6404**	**34.1104**	22.0679	29.2287	37.9311	22.3426	28.5987	50.9506
c5315	30.5597	**21.8695**	43.8268	31.1048	23.0021	56.2116	**25.9356**	23.452	**36.5648**
c6288	35.8091	39.6309	**39.4172**	36.6559	39.2868	40.0324	**34.2615**	**38.4413**	51.6775
c7552	67.5605	74.2026	**73.0008**	73.4326	94.2245	102.8467	**64.2992**	**61.5337**	79.2451

1) `asprin 100 --stats idiag.lp preference.lp "$test" -`
2) `asprin 100 --stats --dom-heur -1 sign idiag.lp preference.lp "$test" -`
3) `asprin 100 --stats --dom-heur 1 level idiag.lp preference.lp "$test" -`
n) Stripped heuristic from `idiag.lp` to use internal heuristics from asprin.

Table 5 Maximal runtimes of asprin with various heuristics. The best result is highlighted with a bold font.

	asprin[1)]			asprin$_s$[2,n)]			asprin$_l$[3,n)]		
circuit	1	2	3	1	2	3	1	2	3
c432	5.426	3.927	3.746	5.361	3.925	3.911	**3.498**	**3.368**	**3.730**
c499	4.595	4.278	4.129	**4.306**	4.602	4.110	4.868	**4.247**	**3.929**
c880	8.099	7.794	6.751	7.990	7.522	6.686	**7.495**	**7.265**	**6.423**
c1355	15.886	**9.646**	8.969	**14.226**	9.926	**8.905**	14.717	64.170	11.622
c1908	22.879	**18.766**	**19.498**	28.714	18.790	28.001	**16.622**	29.100	56.778
c2670	29.583	22.347	**27.956**	28.675	24.659	28.132	**19.662**	**20.250**	28.685
c3540	33.875	**34.000**	**60.765**	**33.066**	34.784	73.620	44.448	38.024	268.888
c5315	53.090	**39.037**	78.912	58.880	43.743	169.542	**42.327**	56.649	**74.325**
c6288	**40.874**	**42.954**	**42.745**	44.732	43.429	49.227	42.546	43.081	186.843
c7552	**73.987**	**90.521**	**115.544**	100.091	177.680	181.315	85.318	117.532	326.509

1) `asprin 100 --stats idiag.lp preference.lp "$test" -`
2) `asprin 100 --stats --dom-heur -1 sign idiag.lp preference.lp "$test" -`
3) `asprin 100 --stats --dom-heur 1 level idiag.lp preference.lp "$test" -`
n) Stripped heuristic from `idiag.lp` to use internal heuristics from asprin.

References

1 Moritz Bayerkuhnlein and Diedrich Wolter. Model-based diagnosis with ASP for non-groundable domains. In Arne Meier and Magdalena Ortiz, editors, *Foundations of Information and Knowledge Systems*, pages 363–380, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-56940-1_20.

2 Gerhard Brewka, James Delgrande, Javier Romero, and Torsten Schaub. A general framework for preferences in answer set programming. *Artificial Intelligence*, 325:104023, 2023. doi:10.1016/j.artint.2023.104023.

3 Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub. asprin: Customizing answer set preferences without a headache. In *AAAI*, pages 1467–1474. AAAI Press, 2015. doi:10.1609/AAAI.V29I1.9398.

4 Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, December 2011. doi:10.1145/2043174.2043195.

5 Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target simulator in fortran. In *Proceedings of the 1985 International Symposium on Circuits and Systems (ISCAS'85)*, pages 695–698, June 1985. Special Session on ATPG and Fault Simulation.

6 Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020. doi:10.1017/S1471068419000450.

7 Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987. doi:10.1016/0004-3702(87)90063-4.

8 Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. Abstract Gringo. *Theory and Practice of Logic Programming*, 15(4-5):449–463, 2015. doi:10.1017/S1471068415000150.

9 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi:10.1017/S1471068418000054.

10 Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, pages 350–356. AAAI Press, 2013. doi:10.1609/AAAI.V27I1.8585.

11 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

12 Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989. doi:10.1016/0004-3702(89)90079-9.

13 Jacques Herbrand. *Recherches sur la théorie de la démonstration.* Doctoral thesis, Faculté des sciences de Paris, 1930.

14 Vladimir Lifschitz. *Answer Set Programming.* Springer, 2019. doi:10.1007/978-3-030-24658-7.

15 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

16 Iulia Nica, Ingo Pill, Thomas Quaritsch, and Franz Wotawa. The route to success - A performance comparison of diagnosis algorithms. In Francesca Rossi, editor, *IJCAI 2013,*

Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, pages 1039–1045. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6597.

17 Iulia Nica and Franz Wotawa. Condiag - computing minimal diagnoses using a constraint solver. In *Proceedings of the 23rd International Workshop on Principles of Diagnosis*, 2012.

18 Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999. doi:10.1023/A:1018930122475.

19 Ingo Pill and Thomas Quaritsch. Rc-tree: A variant avoiding all the redundancy in reiter's minimal hitting set algorithm. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Gaithersburg, MD, USA, November 2-5, 2015*, pages 78–84. IEEE Computer Society, 2015. doi:10.1109/ISSREW.2015.7392050.

20 Liliana Marie Prikler and Franz Wotawa. Replication package for: Faster diagnosis with answer set programming. The replication package contains our implementation of IDIAG and the scripts we used to benchmark them as two gzipped tar archives. It does not include the test data. doi:10.5281/zenodo.13850098.

21 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

22 Roni Stern, Meir Kalech, Shelly Rogov, and Alexander Feldman. How many diagnoses do we need? *Artificial Intelligence*, 248:26–45, 2017. doi:10.1016/j.artint.2017.03.002.

23 Franz Wotawa. A variant of reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001. doi:10.1016/S0020-0190(00)00166-6.

24 Franz Wotawa and David Kaufmann. Model-based reasoning using answer set programming. *Applied Intelligence*, 52(15):16993–17011, 2022. doi:10.1007/S10489-022-03272-2.

FLEX: Fault Localization and Explanation Using Open-Source Large Language Models in Powertrain Systems

Herbert Muehlburger
Institute of Software Technology, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Technology, Graz University of Technology, Austria

Abstract

Cyber-physical systems (CPS) are critical to modern infrastructure, but are vulnerable to faults and anomalies that threaten their operational safety. In this work, we evaluate the use of open-source Large Language Models (LLMs), such as Mistral 7B, Llama3.1:8b-instruct-fp16, and others to detect anomalies in two distinct datasets: battery management and powertrain systems. Our methodology utilises retrieval-augmented generation (RAG) techniques, incorporating a novel two-step process where LLMs first infer operational rules from normal behavior before applying these rules for fault detection. During the experiments, we found that the original prompt design yielded strong results for the battery dataset but required modification for the powertrain dataset to improve performance. The adjusted prompt, which emphasises rule inference, significantly improved anomaly detection for the powertrain dataset. Experimental results show that models like Mistral 7B achieved F1-scores up to 0.99, while Llama3.1:8b-instruct-fp16 and Gemma 2 reached perfect F1-scores of 1.0 in complex scenarios. These findings demonstrate the impact of effective prompt design and rule inference in improving LLM-based fault detection for CPS, contributing to increased operational resilience.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging; Software and its engineering → Software fault tolerance; Computing methodologies → Machine learning; Software and its engineering; Computing methodologies → Natural language processing

Keywords and phrases Fault detection, anomaly detection, powertrain systems, large language models, open-source LLMs

Digital Object Identifier 10.4230/OASIcs.DX.2024.25

Category Short Paper

Supplementary Material *Software (Source Code and Data)*: `https://github.com/muehlburger/dx-2024-flex` [16], archived at `swh:1:dir:e12a10f926844b17fac4162c73f2aa3a3d56b605`

Funding This study was conducted within the framework of the project ARCHIMEDES, receiving funding from the Key Digital Technologies Joint Undertaking (KDT JU) - the Public-Private Partnership for research, development, and innovation under Horizon Europe – and National Authorities under Grant Agreement No 101112295 funded by the European Union and the FFG under Grant Agreement No FO999899377.

1 Introduction

Cyber-physical systems (CPS) integrate hardware and software with their physical environment, forming the backbone of critical infrastructures such as transportation and energy. Faults in CPS can lead to significant operational and safety risks, making robust fault detection and diagnosis systems essential. Advances in deep learning techniques, such as transformers [21] applied to time-series data [22], have significantly improved the ability to monitor these systems. Fault detection can be applied during system development [13] or


© Herbert Muehlburger and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0



real-time operation, where diagnosing faults promptly is crucial for safety and reliability [15]. In our previous work [14], we evaluated the effectiveness of open-source large language models (LLMs) for fault detection in cyber-physical systems, with a focus on battery management systems. The current work builds on these findings by extending the evaluation to additional system components, such as powertrain systems, and introducing novel prompt designs tailored to different types of anomalies. This allows for more comprehensive fault detection and localization across a wider range of systems.

We explore whether LLMs can be effectively applied for fault detection and diagnosis in CPS. Traditional fault detection approaches have primarily relied on machine learning models and deep learning techniques such as autoencoders [2]. While LLMs like BERT [8] and GPT-3 [5] have shown success in tasks such as rule learning [24] and anomaly detection [17], their use in real-time CPS fault diagnosis remains underexplored.

A key consideration in this study is the exclusive use of open-source LLMs. In contrast to commercial models, open-source LLMs ensure transparency, accessibility, and reproducibility – fundamental requirements in scientific research. Open-source models like Mistral 7B, Llama3.1:8b-instruct-fp16, and Gemma 2 offer the advantage of being freely available and adaptable for diverse applications, making them ideal for research purposes. This choice allows researchers to replicate, validate, and extend the findings, which is crucial for building reliable fault detection systems in CPS.

We focus on using retrieval-augmented generation (RAG) [12] to enhance fault detection by integrating external knowledge through search strategies such as maximal marginal relevance (MMR) and cosine similarity. A central component of our approach is the inference of operational rules: LLMs first infer rules from normal CPS behavior before applying them for fault detection. This two-step approach ensures that the models can identify deviations from normal operations and accurately diagnose faults.

Our main contributions are summarised as follows:

- We assess the performance of six open-source models, including Mistral 7B, Gemma 2, Llama3.1:8b-instruct-fp16, and others for fault detection in CPS datasets.
- We propose a two-step process in which LLMs first infer operational rules from normal CPS behavior and then apply these rules to diagnose faults, highlighting the critical role of rule inference in improving detection accuracy.
- We evaluate different search strategies, including maximal marginal relevance (MMR) and cosine similarity, to retrieve relevant information during the fault detection process in the RAG framework.
- We provide a detailed analysis of the models' ability to generalise from inferred rules and accurately diagnose faults in CPS, offering insights into their performance across different datasets.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 outlines the methodology and datasets used. Section 4 presents and discusses the experimental results. Finally, Section 5 concludes the paper and suggests directions for future research.

2 Related Work

Our work is at the intersection of research on a) security in cyber-physical systems and b) machine learning in fault detection, specifically the application of large language models in anomaly detection.

CPS play a critical role in various essential infrastructure sectors, such as manufacturing and smart grids [7]. Ensuring the security of CPS involves fault detection mechanisms to maintain operational integrity. Previous studies have investigated fault detection in CPS using traditional machine learning techniques, highlighting both achievements and constraints [18]. However, there is a research gap on the use of LLMs for anomaly detection in CPS, particularly in fault detection scenarios [11].

Recent discussions have surfaced about the use of open-source LLMs in cybersecurity, offering theoretical insights and initial findings on their potential applications [11]. Furthermore, research has examined textual encoding and its impact on the performance of machine learning models, especially in contexts relevant to CPS security [11]. Recognising this research gap, this work aims to explore the application of open-source LLMs to fault detection in CPS, presenting avenues for future investigations [11].

In the domain of fault detection and isolation, researchers have proposed robust methods for fault diagnosis in CPS, focusing on fault detection, isolation, identification, and risk assessment processes [4]. Furthermore, studies have explored anomaly detection in CPS using Bayesian networks to establish cause-and-effect relationships between cyber components and physical systems [11]. These integrated approaches are crucial to effectively detect anomalies and distinguish between cyber attacks and physical faults in CPS [11].

Previous studies, including our own work [14], have explored the application of LLMs to fault detection in CPS, with a specific focus on battery management systems. In that work, we demonstrated the efficacy of open-source LLMs such as Mistral for fault detection using retrieval-augmented generation (RAG) techniques. However, that study primarily dealt with fault detection in isolated systems (i.e., the battery management system), whereas this work expands the focus to more complex, multi-component systems like powertrains. Additionally, this paper introduces new prompt designs optimized for fault localization in systems with interdependent components.

The research discussed in this section highlights the importance of fault detection to ensure the security and reliability of CPS. The application of machine learning, specifically LLMs, provides opportunities to further enhance the field of CPS security and fault detection.

3 Methodology

Building on our previous evaluation of LLMs for fault detection in battery management systems [14], we adapted the methodology to handle the more complex fault localization requirements of powertrain systems. While the initial study focused on detecting anomalies in a single system component, this work extends the approach by designing new prompts that capture interactions between multiple components (battery, H-Bridge, and motor) to improve the accuracy and explanatory power of fault detection. In this study, we implement a fault detection system for powertrain components using a combination of LLMs, embeddings, and vector search. Figure 1 visualizes FLEX, our proposed approach to using LLMs for fault detection. In this section, we further describe the dataset used in our experiments. We go into more detail about the data collection procedure, the data preprocessing steps, the model selection, and the experimental setup. Finally, we highlight some limitations of our work.

3.1 Datasets

We use two datasets that represent normal and faulty behavior of two different CPS. The first CPS is a battery management system from an electrified vehicle, which we refer to as the "battery" dataset. The second dataset, called "powertrain," contains simulated time-series data of various powertrain components, such as battery current, voltage, motor speed, and H-Bridge mode.

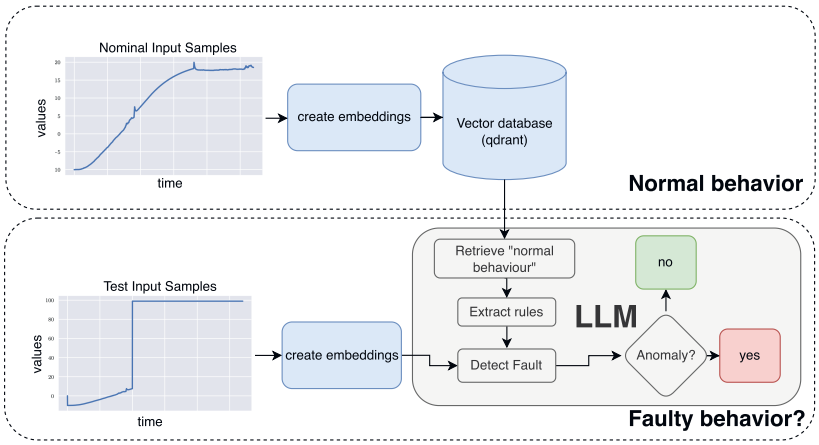


Figure 1 The FLEX approach for fault detection. Nominal (normal) input samples are text encoded and embedded into a vector database. Test samples undergo the same encoding and embedding process. A Retrieval-Augmented Generation (RAG) mechanism is used to query the vector database for similar normal behavior samples. The LLM extracts rules from these normal samples and applies them to detect anomalies in the test samples.

Battery Dataset. The battery dataset is a multivariate time-series obtained from a simulated industrial battery management system. It contains 90 minutes (5,400.5 seconds) of continuous readings across 13 different signals, including temperature sensors, power usage, velocity, and airflow from cooling systems, amounting to 10,801 samples.

We use the training dataset, which contains no anomalies, to represent the normal behavior of the system in a semi-supervised setting. This training data is stored in a vector database for Retrieval-Augmented Generation (RAG) purposes, allowing context retrieval during LLM prompting. A separate test dataset containing anomalies, also with 10,801 samples, is used for model evaluation. The experimental setup is further detailed in Section 3.4.

Powertrain Dataset. The powertrain dataset is derived from a simplified simulation model of an electrified powertrain system. [23] This system consists of a battery, an H-bridge, an electric motor, and a load. The powertrain model is used to generate both correct and faulty behavior data for different loads and faults in the system.

The training dataset includes normal behavior under various load conditions, while the test dataset introduces specific faults in the battery, H-bridge, and motor components. Faults are introduced at various time points in the simulation. For example, faults such as an empty battery or malfunctioning H-bridge switches are simulated. The dataset includes battery voltage, current, motor rotational speed, and H-bridge mode over time for both correct and faulty behaviors, allowing for precise fault diagnosis.

3.2 Data Pre-processing

The pre-processing of our dataset is a crucial step toward leveraging the capabilities of pre-trained LLMs for anomaly detection within CPS. This process involves transforming raw data into a format that is more suitable for analysis by these models. Our pre-processing pipeline consists of several stages outlined below and in Figure 2:

1. **Rounding:** Each numerical value in the dataset is rounded to two decimal places. This simplification helps in standardising the data format and reduces noise in the data that is insignificant to the model's performance.

2. **Text Encoding:** We convert the dataset from numerical to textual representation to align with the input format expected by LLMs. This encoding process involves:
   a. Prefixing each value with its corresponding column name, for example, converting a temperature reading column to "temperature_5_deg".
   b. Appending the text " is " followed by the rounded value of the reading, resulting in entries such as "temperature_5_deg is -9.99".
   c. Convert every number into a word representation resulting in entries such as "temperature_five_deg is minus nine point nine nine".

This textual transformation helps in aligning the dataset with the linguistic input format preferred by LLMs, thereby facilitating more effective anomaly detection by enabling the models to process and analyze the data within a contextually richer format.

3.3 Model Selection and Implementation Strategy

In this study, we deliberately opted for open-source LLMs instead of proprietary options like ChatGPT. Our goal was to ensure reproducibility and openness, making the models we used accessible and easy to replicate for future research. The specific LLMs selected for this study include *Llama3* [9], *Llama3.1:8b-instruct-fp16* [9], *Gemma 2* [20], *Phi3:14b* [1], *Mistral* [10], and *Mistral NeMo* [3]. These models were specifically chosen for their capabilities in anomaly detection within CPS. By selecting a diverse range of open-source models, we were able to explore the strengths and limitations of each in the context of fault detection in CPS. This approach allowed us to evaluate how well these models can handle the complexities of CPS data and their effectiveness in identifying potential faults.

To enhance model performance, we integrated these LLMs into a Retrieval Augmented Generation (RAG) framework. This framework enabled the models to access relevant, anomaly-free operational data from a vector store, improving their ability to detect deviations from normal behavior. The underlying hypothesis was that by providing LLMs with precise representations of normal system operations, they would be better equipped to identify faults.

3.4 Experimental Setup

Our experimental setup[1], as illustrated in Figure 1, uses data represented as multivariate time series in CSV files. Each row contains multiple sensor readings and represents one training sample. We implemented a Retrieval Augmented Generation (RAG) [12] application that uses training samples to provide a relevant context for LLMs. The training data are transformed into an embedding and stored in a vector database. Before prompting the LLM, we query the vector database to find values similar to the test data. Retrieval-augmented generation (RAG) models enhance the capabilities of generative models by incorporating relevant information extracted from an external knowledge base during the generation process. A critical step in this approach is the selection of relevant documents or text snippets from the knowledge base, which is achieved using similarity measures. We used a retriever to fetch relevant documents from the vector store using similarity measures, such as cosine similarity and Maximal Marginal Relevance (MMR) [6]. The retrieved documents were used as context for the LLM, which then processed each measurement from the test dataset, applying the previously inferred rules. The LLM was executed in a pipeline, taking the test data and context as input and producing a fault diagnosis for each time series point.

[1] Our code is available open-source on Zenodo: `https://doi.org/10.5281/zenodo.13879936`

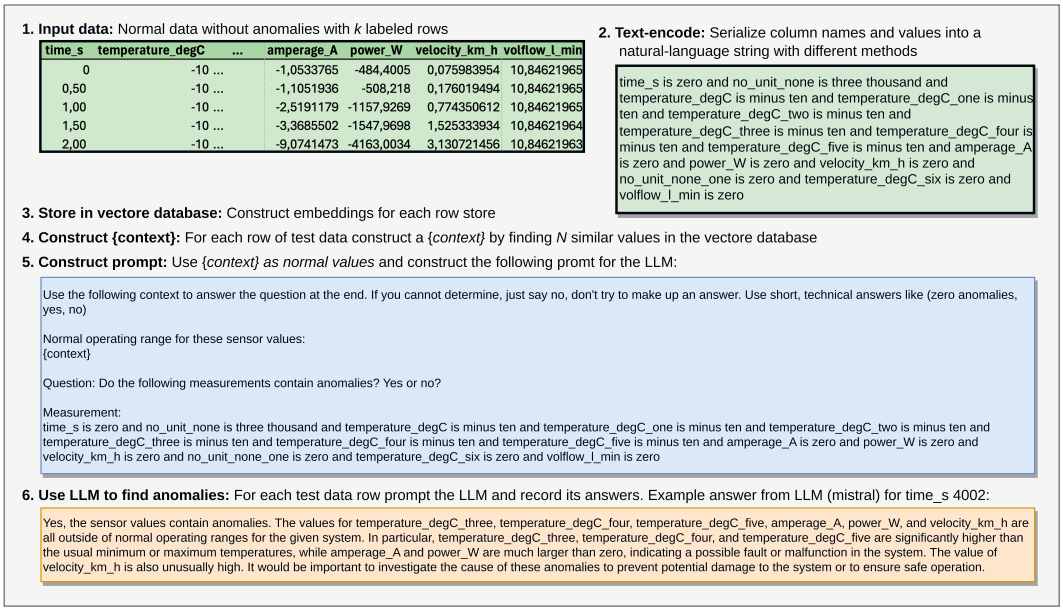


Figure 2 Schematic representation of the anomaly detection workflow using the LLM Mistral for the "battery" dataset. The process begins by ingesting time series data (1), which is transformed into descriptive natural language through text-encoding feature names and their corresponding values (2). These textual representations are embedded using the *SentenceTransformer* library and stored in a vector database (3). The LLM retrieves contextually relevant data by querying this database with test data (4). The retrieved context is then used to prompt the LLM for anomaly detection (5), where the LLM assesses each row by comparing sensor readings with normal operational ranges. The model identifies anomalies, assigns an anomaly rating, and provides explanations for any deviations (6). The example shows a detailed response from Mistral 7B analysing time step 4002, highlighting the model's ability to detect and explain faults in the "battery" dataset.

3.4.1 Application in Retrieval-Augmented Generation

In the context of RAG, similarity measures, such as cosine similarity and MMR, are used to identify documents or text snippets within the knowledge base that are most relevant to the given query. Specifically, the query and each candidate document are represented as vectors in a high-dimensional space, where each dimension corresponds to a feature represented as an embedding. The similarity between the query vector and each document vector is calculated, and the documents are ranked based on their similarity scores. The highest-scoring documents are considered the most relevant and are subsequently used to augment the generative process, generating more informed and contextually relevant text. Cosine similarity and MMR represent different similarity measures, leading to different rankings.

3.4.2 Fault Detection Prompt Design

Initially, we designed a simple prompt for the battery system, focusing on voltage stability as the main indicator of faults. However, when we applied this prompt to the powertrain system, it did not produce satisfactory results. The powertrain system is more complex, involving multiple interdependent components like the battery, H-Bridge, and motor, each with distinct fault patterns. Using the same battery prompt for powertrain would not enable the LLM to effectively capture and differentiate these complex fault scenarios.

Listing 1 Prompt Template for "battery" dataset. The prompt contains a {context} with samples that do not contain anomalies retrieved from the vector database. We search for the ten most similar entries in the vector database compared to the test data {measurement}.

```
Use the following context to answer the question at the end. If you cannot determine, just say no, don't try to make up an answer. Use short, technical answers like (zero anomalies, yes, no). Write a response that appropriately completes the request.

Context: Normal operating range:
{context}

Question: Do the following measurements contain anomalies? Yes or no?

Test Measurement:
{measurement}

Provide an anomaly rating between 0—100 based on the following criteria:

1) Values are in a reasonable range of the provided context.
2) Values do not contain anomalies.
3) Values are consistent with the given context.

Check *very carefully* with the given ranges before giving your rating.

Answer:
— Anomaly: e.g. yes, no
— Anomaly—Rating:
— Explanation:
```

We also chose not to use the more complex powertrain prompt for the battery system, as it emphasises interactions between components, which are not necessary for detecting battery faults. The battery system relies primarily on voltage and current for fault detection, making a simpler approach more efficient. Applying the more detailed powertrain prompt to the battery system would add unnecessary complexity and reduce efficiency, but could be explored in future work.

Given these differences, we constructed a two-part prompt specifically for the powertrain system. In the first part, the LLM infers operational rules from normal behavior, identifying expected ranges for key variables like battery current, voltage, motor speed, and H-Bridge mode. These rules are strictly based on time-series data and do not assume any values outside the provided context. In the second part, the LLM uses the inferred rules to detect faults by comparing test data against the established normal behavior. Faults are only reported when there is a clear deviation from normal operations, ensuring a high level of precision in the fault detection process. This strict two-step approach helped improve fault detection accuracy in the powertrain system. We used the prompt template from Listing 1 for the "battery" dataset and the prompt templates from Listings 2 and 3 for the "powertrain" dataset. These prompts were implemented using the Langchain[2] Python framework and LLMs were accessed locally through Ollama[3].

[2] H. Chase. "Langchain." python.langchain.com. `https://python.langchain.com/docs/get_started/introduction` (accessed Sep. 05, 2024).
[3] Ollama. "Ollama v0.3.9." `https://ollama.com` (accessed Sep. 05, 2024).

Listing 2 System Prompt template for "powertrain" dataset. The system prompt provides strict instructions for rule inference and fault detection, to prevent hallucination of faults.

```
Your goal is to evaluate the overall status of the battery based on the provided data. The battery status should only be "OK" or "Empty." Do not mention individual sensor values such as voltage or current in the final report. The output should be in a concise, structured format and only focused on the battery as a whole component.

### Battery Fault Models:
- OK: The battery is functioning as expected.
- Empty: The battery provides no voltage (Battery Voltage = 0).

### For inferred rules:
- You must infer rules based on the provided time series data.
- Rules should represent the overall battery behavior as either "OK" or "Empty."

Example format for inferred rules:
- Battery: OK (the battery voltage remains stable)

### For fault detection:
- You MUST NOT infer faults that are not represented by the data. Report faults only if the data clearly shows that the battery provides no voltage (Battery Empty).

Output format must strictly be:
component,fault,reason
battery,OK/Empty,Explanation
```

Listing 3 User Prompt template for "powertrain" dataset. The user prompt guides the process of rule inference and fault detection based on time series data.

```
# Part 1: Infer Rules from Normal Battery Behavior

Use the following time series data to infer the overall behavior of the battery as either "OK" or "Empty."

# Input Data:
{context}

---

# Part 2: Battery Fault Detection Based on Inferred Rules

Now, apply the inferred rules strictly to the following test data to detect if the battery is "OK" or "Empty."

# Battery Fault Models:
- OK: The battery is functioning as expected.
- Empty: The battery provides no voltage (Battery Voltage = 0).

# Input Data:
{measurement}

# Task:
1. Based ONLY on the inferred rules, determine whether the battery is "OK" or "Empty."
2. Report the results in the following format:
component,fault,reason
battery,OK/Empty,Explanation
```

3.5 Evaluation Metrics

In this work, we assess the performance of open-source LLMs in diagnosing faults in CPS using two datasets: "battery" and "powertrain." Our evaluation is based on two key experiments: the first evaluates models Mistral and Llama3 on 100 samples from the "battery" dataset, while the second evaluates the models Mistral, Mistral NeMo, Llama3.1:8b-instruct-fp16, Gemma 2, and Phi3:14b on 10 samples from the "powertrain" dataset.

Seliya et al. [19] proposed several metrics for classifier evaluation. For our study, we selected precision, recall, and F1-score to compare the performance of the models in fault detection tasks. These metrics are defined as follows:

Precision measures the proportion of correctly identified faults (true positives) out of all instances flagged as faulty, i.e.: Precision $= \frac{TP}{TP+FP}$. High precision is crucial in fault detection, as it minimises false positives, preventing unnecessary maintenance or downtime.

Recall, or sensitivity, measures the proportion of actual faults that were correctly identified, i.e.: Recall $= \frac{TP}{TP+FN}$. High recall ensures that most faults are detected, even if some false positives are generated.

The **F1-score** is the harmonic mean of precision and recall, balancing the two, i.e.: $F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$. A high F1-score indicates that the model achieves a good balance between precision and recall, which is essential for accurate fault detection, managing both false positives and false negatives effectively.

4 Results

As demonstrated in our previous work [14], open-source LLMs such as Mistral perform exceptionally well in detecting anomalies in battery management systems. However, our results indicate that applying the same prompt structure to powertrain systems does not yield similarly performance. Therefore, this paper introduces a new two-part prompt designed specifically for powertrain systems, leading to improved fault detection performance. This section presents the performance of several LLMs on anomaly detection in CPS, considering the role of prompt design in shaping model performance. The experiments were conducted using two datasets: "battery" and "powertrain," with different prompt structures for each. The models were evaluated based on F1-score, Precision, and Recall, as shown in Tables 1 and 2.

4.1 Effect of Prompts on Model Performance

The prompts used for each dataset had a significant impact on the performance of the models. In the "battery" dataset, the prompt (Listing 1) was straightforward, asking the LLMs to identify anomalies by comparing test measurements with normal system behavior retrieved from the vector database. In contrast, the "powertrain" dataset utilized more complex prompts (Listings 2 and 3) that required the LLMs to first infer rules from normal system behavior before applying them to detect faults. This two-step process introduced additional cognitive complexity, which likely influenced the models' performance differently across the datasets.

4.2 Battery Dataset Results

The "battery" dataset prompt provided a clear context of normal behavior and asked the LLMs to determine whether test measurements contained anomalies based on this context. As shown in Table 1, *Mistral* consistently outperformed *Llama3* across different configurations

Table 1 Performance of Large Language Models (LLMs) on anomaly detection tested using 100 samples from the "battery" dataset. The table evaluates two models, Mistral and Llama3, across two retrieval strategies: Maximal Marginal Relevance (MMR) and cosine similarity. The models were assessed with varying numbers of retrieved results (ranging from 1 to 1000), with performance measured using F1-score, precision, and recall. Notably, Mistral consistently achieved near-perfect scores (F1=0.99, Precision=1.00, Recall=0.98) across multiple configurations, while Llama3 showed lower precision when fewer results were retrieved, despite maintaining perfect recall.

Model	Strategy	# Results	F1	Precision	Recall
llama3	mmr	1	0.68	0.52	1.00
llama3	similarity	1	0.68	0.52	1.00
mistral	similarity	1000	0.81	0.69	1.00
mistral	similarity	500	0.86	0.77	0.98
mistral	similarity	10	0.96	0.94	0.98
llama3	mmr	5	0.98	1.00	0.96
mistral	mmr	1000	0.98	0.98	0.98
mistral	mmr	500	0.98	0.98	0.98
llama3	mmr	10	**0.99**	1.00	0.98
mistral	mmr	1	**0.99**	1.00	0.98
mistral	similarity	1	**0.99**	1.00	0.98
mistral	mmr	10	**0.99**	1.00	0.98
mistral	mmr	5	**0.99**	1.00	0.98
mistral	similarity	5	**0.99**	1.00	0.98

and retrieval strategies (MMR and similarity). *Mistral* achieved an F1-score of 0.99 when retrieving 1, 5, or 10 documents, with near-perfect precision (1.00) and recall (0.98). This high performance can be attributed to the direct nature of the prompt, which simply asked the models to compare test data with retrieved normal data without needing to infer additional rules.

In contrast, *Llama3* showed more variability in performance, especially when fewer documents were retrieved. The best F1-score (0.99) for *Llama3* was observed when 10 documents were retrieved with the MMR strategy, yielding a precision of 1.00 and recall of 0.98. However, when only 1 document was retrieved, the F1-score dropped to 0.68, with a precision of 0.52. This variability suggests that *Llama3* relies more heavily on sufficient contextual information to perform optimally. The straightforward nature of the battery prompt worked in favor of *Mistral*, while *Llama3* struggled with limited context, leading to more false positives.

4.3 Powertrain Dataset Results

The "powertrain" dataset employed a more sophisticated two-step prompt, where the LLMs first had to infer rules from the context of normal behavior and then apply these rules to detect faults in the test data. This rule inference process significantly impacted the models' performance, as it required a deeper understanding of normal behavior and strict adherence to the inferred rules before making fault predictions.

As detailed in Table 2, *Mistral-NeMo* struggled with this complexity, achieving an F1-score of only 0.33, with perfect precision (1.00) but very low recall (0.20). The need to infer rules and then apply them to the test data may have caused *Mistral-NeMo* to miss many true anomalies, leading to poor recall. This highlights the model's difficulty in managing tasks that require both rule inference and anomaly detection.

Table 2 Comparative performance of various Large Language Models (LLMs) on anomaly detection, tested with 10 samples from the "powertrain" dataset. The table evaluates models such as Mistral-Nemo, Phi3:14b, Gemma2, Llama3.1:8b-instruct, and Mistral across two retrieval strategies: Maximal Marginal Relevance (MMR) and cosine similarity. The performance is measured using F1-score, precision, and recall. Notably, models like Gemma2, Llama3.1:8b-instruct, and Mistral achieved perfect performance (F1=1.00, Precision=1.00, Recall=1.00), while Mistral-Nemo struggled with lower recall (0.20) despite perfect precision, indicating difficulties in detecting all anomalies in the test samples.

Model	Strategy	# Results	F1	Precision	Recall
mistral-nemo	mmr	10	0.33	1.00	0.20
mistral-nemo	similarity	10	0.33	1.00	0.20
phi3:14b	mmr	10	0.89	1.00	0.80
phi3:14b	similarity	10	0.89	1.00	0.80
gemma2	mmr	10	**1.00**	1.00	1.00
gemma2	similarity	10	**1.00**	1.00	1.00
llama3.1:8b-instruct-fp16	mmr	10	**1.00**	1.00	1.00
llama3.1:8b-instruct-fp16	similarity	10	**1.00**	1.00	1.00
mistral	mmr	10	**1.00**	1.00	1.00
mistral	similarity	10	**1.00**	1.00	1.00

In contrast, *Phi3:14b* performed much better, achieving an F1-score of 0.89 with both MMR and similarity strategies, balancing perfect precision (1.00) with a strong recall of 0.80. This model was more adept at handling the rule inference step, allowing it to maintain a good balance between detecting true anomalies and avoiding false positives.

Both *Gemma 2* and *Llama3.1:8b-instruct-fp16* performed flawlessly in the "powertrain" dataset, achieving perfect F1-scores (1.00) across all retrieval strategies. These models successfully inferred rules and applied them to detect anomalies with no errors, suggesting that they are particularly well-suited for tasks that involve complex, multi-step prompts like the one used in the "powertrain" dataset.

4.4 Implications of Rule Inference on Model Behavior

The rule inference step in the "powertrain" dataset introduced an additional layer of complexity that had clear implications for model performance. Models that were able to handle this step effectively, such as *Gemma 2*, *Llama3.1:8b-instruct-fp16*, and *Phi3:14b*, performed well in both precision and recall, demonstrating their ability to generalize from inferred rules and apply them accurately to detect faults. This process required models to not only understand normal behavior but also apply that understanding in a precise manner to detect deviations.

On the other hand, models like *Mistral-NeMo* struggled with this process, leading to a significant drop in recall. This suggests that models less capable of managing multi-step tasks or those that struggle with rule inference may fail to detect all anomalies, particularly in complex scenarios. This limitation highlights the importance of choosing models that can effectively handle not only anomaly detection but also rule-based reasoning tasks in more sophisticated use cases.

4.5 Cross-Dataset Comparison

Across both datasets, *Mistral* proved to be the most consistent performer, achieving strong results in the simpler "battery" prompt and perfect results in the more complex "powertrain" prompt. *Llama3* displayed high sensitivity to the number of retrieved documents in the "battery" dataset, performing well when given sufficient context but struggling when that context was limited. However, the *Llama3.1:8b-instruct-fp16* variant performed perfectly in the "powertrain" dataset, suggesting that specific configurations of *Llama3* are better suited to complex, multi-step tasks.

The performance of *Mistral-NeMo* in the "powertrain" dataset illustrates how rule inference can overwhelm models that are not well-equipped for this type of reasoning. In contrast, models like *Gemma 2* and *Phi3:14b* thrived in this context, handling both the rule inference and fault detection steps effectively.

4.6 Key Insights

- **Prompt Complexity Influences Results**: The simpler prompt in the "battery" dataset led to more consistent performance, while the complex rule inference prompt in the "powertrain" dataset revealed varying levels of model proficiency. Models that managed rule inference well performed better in the "powertrain" dataset.
- **Mistral's Robustness**: *Mistral* maintained high performance across both datasets, demonstrating its ability to manage both straightforward and more complex tasks.
- **Rule Inference Sensitivity in Llama3**: While *Llama3* performed well with sufficient context in the "battery" dataset, its *Llama3.1:8b-instruct-fp16* variant excelled in the more complex "powertrain" task, suggesting its suitability for tasks that require rule-based reasoning.
- **Mistral-NeMo's Struggle with Complexity**: *Mistral-NeMo* struggled in the "powertrain" dataset due to the complexity of rule inference, indicating the need for more tuning or alternative approaches when handling multi-step tasks.

4.7 Insights from Model Responses

Analysis of the models' generated responses offers insights into their decision-making processes. For example, Mistral's responses provide a structured output comprising a Boolean flag for anomaly detection, an anomaly rating quantifying detection certainty, and a textual explanation justifying the decision. Interestingly, an examination of Mistral's reasoning reveals occasional inaccuracies in its assessments. The LLM detected an anomaly and provided an explanation arguing that a value of -1.42 is not part of the range of -10 to 0.8. This argument is clearly wrong. Interestingly, if we provide the same answer to the same LLM in a second round, it notices that the answer is not correct and the value is within the given range. This discrepancy underscores the importance of iterative validation and refinement in improving the interpretability and reliability of the model in anomaly detection tasks.

The decision-making process in battery fault detection also showcases the nuances in model behavior. Let us illustrate how Llama3.1:8b-instruct-fp16 generates detailed, multi-step diagnostics based on time series and test data, inferring the state of the battery as either "OK" or "Empty" considering two queries. In the first query, the model analyzes a scenario where the battery's voltage remains stable at 14, with no signs of discharge or abnormal behavior. By applying these inferred rules to the test data, Llama3.1:8b-instruct-fp16 concludes that the battery is functioning normally, providing stable voltage, and classifies it as "OK." This

demonstrates the model's ability to detect stable battery behavior based on consistent voltage readings and an absence of current flow. In another query, Llama3.1:8b-instruct-fp16 identifies a battery fault. The model, after inferring the same rules from normal battery behavior, detects that the voltage has dropped to zero in the test data. Based on this observation, the model correctly concludes that the battery is "Empty," citing the zero voltage as the primary reason for the fault. These example illustrates the model's application of learned rules to identify faults when deviations from normal behavior occur.

5 Conclusion & Future Work

This paper explored the use of open-source LLMs for fault detection in CPS, focusing on battery management and powertrain datasets. Models like *Mistral 7B* and *Llama3.1:8b-instruct-fp16* performed well, with *Gemma 2* and *Phi3:14b* achieving perfect F1-scores in complex scenarios. The results highlight the importance of prompt design and rule inference for accurate fault detection. Challenges remain, as some models, like *Mistral-NeMo*, struggled with more complex tasks, underscoring the need for model selection tailored to task complexity. Future work should explore more sophisticated fault diagnosis tasks and adaptive retrieval methods that further enhance LLM performance in dynamic CPS environments.

Ethical Statement

This research does not involve any ethical concerns or conflicts of interest.

References

1 Marah Abdin, Jyoti Aneja, and Hany Awadalla. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone, August 2024. doi:10.48550/arXiv.2404.14219.

2 Abhaya Abhaya and Bidyut Kr. Patra. An efficient method for autoencoder based outlier detection. *Expert Systems with Applications*, page 118904, September 2022. doi:10.1016/j.eswa.2022.118904.

3 Mistral AI-Team. Mistral NeMo. https://mistral.ai/news/mistral-nemo/, July 2024.

4 Joseph Osedome Boi-Ukeme. A Robust Discrete Event Method for the Design of Cyber-Physical Systems, January 2023.

5 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. arXiv:2005.14165, doi:10.48550/arXiv.2005.14165.

6 Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336, Melbourne Australia, August 1998. ACM. doi:10.1145/290941.291025.

7 Marco Cook, Cory Paterson, Angelos K. Marnerides, and Dimitrios Pezaros. Anomaly Diagnosis in Cyber-Physical Systems. In *ICC 2022 - IEEE International Conference on Communications*, pages 5445–5450, May 2022. doi:10.1109/ICC45855.2022.9838968.

8 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. arXiv:1810.04805, doi:10.48550/arXiv.1810.04805.

9 Abhimanyu Dubey, Abhinav Jauhri, and Pandey. Llama 3 Herd of Models, August 2024. arXiv:2407.21783.

10 Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, October 2023. doi:10.48550/arXiv.2310.06825.

11 Sudha Krishnamurthy, Soumik Sarkar, and Ashutosh Tewari. Scalable Anomaly Detection and Isolation in Cyber-Physical Systems Using Bayesian Networks. In *ASME 2014 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, December 2014. doi:10.1115/DSCC2014-6365.

12 Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, M. Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *ArXiv*, May 2020.

13 Herbert Muehlburger and Franz Wotawa. A Passive Testing Approach using a Semi-Supervised Intrusion Detection Model for SCADA Network Traffic. In *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 42–47, August 2022. doi:10.1109/AITest55621.2022.00015.

14 Herbert Muehlburger and Franz Wotawa. FaultLines: Evaluating the efficacy of open-source large language models for fault detection in cyber-physical systems. In *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*, Shanghai, China, July 2024. IEEE Computer Society.

15 Herbert Muehlburger and Franz Wotawa. Making Systems Fail-Aware: A Semi-Supervised Machine Learning Approach for Identifying Failures by Learning the Correct Behavior of a System. *IFAC-PapersOnLine*, 58(4):7–12, January 2024. doi:10.1016/j.ifacol.2024.07.185.

16 Herbert Mühlburger and Franz Wotawa. muehlburger/dx-2024-flex. Software, swhId: swh:1:dir:e12a10f926844b17fac4162c73f2aa3a3d56b605 (visited on 2024-11-14). URL: https://github.com/muehlburger/dx-2024-flex.

17 Jonathan Pan, Swee Liang Wong, and Yidi Yuan. RAGLog: Log Anomaly Detection using Retrieval Augmented Generation, November 2023. doi:10.48550/arXiv.2311.05261.

18 Vasso Reppa, Marios M. Polycarpou, and Christos G. Panayiotou. Distributed Sensor Fault Diagnosis for a Network of Interconnected Cyberphysical Systems. *IEEE Transactions on Control of Network Systems*, 2(1):11–23, March 2015. doi:10.1109/TCNS.2014.2367362.

19 Naeem Seliya, Taghi M. Khoshgoftaar, and Jason Van Hulse. A Study on the Relationships of Classifier Performance Metrics. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 59–66, November 2009. doi:10.1109/ICTAI.2009.25.

20 Gemma Team, Morgane Riviere, and Shreya Pathak. Gemma 2: Improving Open Language Models at a Practical Size, August 2024. doi:10.48550/arXiv.2408.00118.

21 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv:1706.03762.

22 Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in Time Series: A Survey, May 2023. arXiv:2202.07125.

23 Franz Wotawa, David Kaufmann, Adil Amukhtar, Iulia Nica, Florian Klück, Hermann Felbinger, Petr Blaha, Matus Kozovsky, Zdenek Havranek, and Martin Dosedel. *Real-Time Predictive Maintenance – Model-Based, Simulation-Based and Machine Learning Based Diagnosis*, pages 63–81. River Publishers, New York, 1 edition, September 2022. doi:10.1201/9781003337232-7.

24 Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. Large Language Models can Learn Rules, October 2023. doi:10.48550/arXiv.2310.07064.

Hyperplanes Based Zonotopic Contractor

Rahma Bengamra
LAAS-CNRS, Université de Toulouse, UPS, 7 avenue du Colonel Roche, 31400 Toulouse, France

Soheib Fergani
LAAS-CNRS, Université de Toulouse, UPS, 7 avenue du Colonel Roche, 31400 Toulouse, France

Carine Jauberthie
LAAS-CNRS, Université de Toulouse, UPS, 7 avenue du Colonel Roche, 31400 Toulouse, France

Abstract

This article introduces a novel method for constructing a "zonotopic contractor" based on hyperplane properties. While zonotopes, a special class of polytopes, offer computational advantages due to their symmetric matrix representation, they are not closed under intersection, often necessitating over-approximations that lead to conservatism in practical applications. The proposed contractor addresses this issue by providing a more efficient solution for approximating zonotope intersections, reducing conservatism. This method generates a new zonotope that closely covers the intersection of two zonotopes. The reliability and effectiveness of the proposed approach are demonstrated through simulations on a bicycle model, showing potential benefits in safety-critical applications like autonomous driving, where precise uncertainty management is crucial for decision-making and control.

2012 ACM Subject Classification Mathematics of computing → Continuous mathematics

Keywords and phrases Contractors, Zonotopes, Hyperplanes

Digital Object Identifier 10.4230/OASIcs.DX.2024.26

Category Short Paper

1 Introduction

With the increasing complexity of modern automated systems, precise state estimation has become essential for effective monitoring and control. Accurate state knowledge is crucial for ensuring the smooth operation of systems, such as robotics, autonomous vehicles, and aerospace platforms, where real-time decision-making is required. This precision is achieved by processing sensor data, which is often noisy or incomplete, through advanced mathematical tools known as filters.

Filters are designed to estimate key system variables like position and velocity, even in the presence of uncertainty. Depending on the nature of the data and assumptions about uncertainty, different types of filters are employed. For instance, probabilistic filters, such as the Kalman filter [6], use probability distributions to model uncertainty and are particularly effective in linear systems subjected to Gaussian noise. However, in many cases, statistical assumptions do not hold or are unavailable, necessitating alternative methods. In such scenarios, set-membership methods are often preferred, representing uncertainty through compact sets without relying on probabilistic models. These sets can be represented by various geometric shapes in the state space, each with distinct strengths and limitations:

- Ellipsoids [13] [3], which provide a compact and well-suited representation of sets but are not closed under Minkowski sum, limiting their use in some applications.
- Polytopes [14], which offer flexibility and accurate representations of uncertainty, though they are computationally intensive due to their complex vertex and facet representation.
- Intervals [12] [10], which simplify calculations by bounding each variable independently, but are highly conservative and may not capture the true uncertainty effectively.


© Rahma Bengamra, Soheib Fergani, and Carine Jauberthie;
licensed under Creative Commons License CC-BY 4.0



- Zonotopes [7] [8] [1] [9], a special class of polytopes, strike a balance between simplicity and computational efficiency. They can be represented compactly using matrices, making operations faster and less conservative than intervals.

Despite their advantages, zonotopes face a significant limitation: they are not closed under intersection. This means that the intersection of two zonotopes does not necessarily yield another zonotope, complicating the representation of uncertainty in systems requiring intersection operations. As a result, overapproximation is often used, which introduces conservatism and reduces the precision of state estimation.

To overcome this challenge, we propose a novel "zonotopic contractor" that leverages hyperplane properties to approximate the intersection of two zonotopes while minimizing conservatism. The proposed contractor generates a new zonotope that closely covers the intersection, significantly improving the accuracy of set-based methods in systems where zonotope intersections are required. This method is designed to maintain the computational simplicity of zonotopes while addressing their limitations, offering a more efficient solution for state estimation in complex systems. We validate the performance of this zonotopic contractor through simulations using a bicycle model, demonstrating its effectiveness in reducing conservatism and enhancing reliability.

2 Preliminaries

▶ **Definition 1** (Zonotopes). *A zonotope is a type of geometric shape that is convex and symmetric. To create it, we start with a center point $c \in \mathbb{R}^n$ and then extend one or more vectors outward from this center in different directions. These vectors, which we call generating vectors and denote as $g_i \in \mathbb{R}^n : i \in 1, ..., d$, where n is the dimention and d represents the number of vectors, determine the shape of the zonotope. The zonotope's order $\rho = d/n$, represents a dimensionless measure of the representation size. i.e., This provides a relative measure of the complexity or dimension of the space covered by the zonotope. The higher this order, the larger and more complex the zonotope becomes.*

For each generator vector, we also include another vector pointing in the opposite direction. This pairing creates segments, i.e., forming a set of points lying within the interval I defined by:

$$I_i = c_i + [-1, 1]g_i = \{c_i + \alpha g_i : \alpha \in [-1, 1]\}, \quad \text{for } i = 1, \ldots, d. \tag{1}$$

After obtaining the intervals, we compute their Minkowski sum. Which is defined by :

$$I_1 + I_2 := \{x + y : x \in I_1, y \in I_2\} \tag{2}$$

So indeed, the zonotope is the Minkowski sum of d segments in $\mathbb{R}^n$.

Now that we have defined zonotopes, let's imagine a hypercube formed by the intervals I_i where i ranges from 1 to d. According to the definition we provided for a zonotope, we can construct a zonotope from these intervals by applying the Minkowski sum as :

$$Z = \sum_{i=1}^{d} I_i = \sum_{i=1}^{d} c_i + [-1, 1]g_i = \sum_{i=1}^{d} c_i + \sum_{i=1}^{d} [-1, 1]g_i = c + G * [-1, 1]^d \tag{3}$$

$$Z = \{x \in \mathbb{R}^n : x = c + G * \alpha^d : \alpha \in [-1, 1]\} \tag{4}$$

Therefore, the zonotope Z is defined by the new center c, which is the sum of the centers c_i of the intervals I_i and the generatrice matrix G such that $G = [g_1, ..., g_d]$.

Hence, another definition of a zonotope is that it is an affine transformation of the interval box of dimension $d \geq n$.

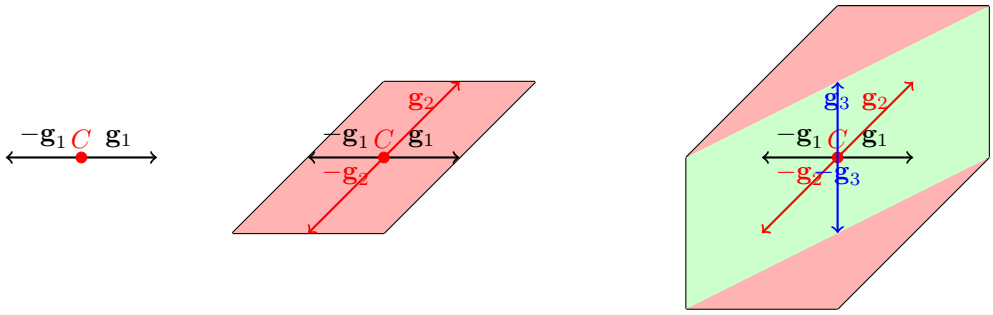


Figure 1 Illustration of the steps involved in constructing a zonotope from three vectors.

We have a matrix G composed of the following vectors: $g_1 = \begin{bmatrix}1\\0\end{bmatrix}$, $g_2 = \begin{bmatrix}1\\1\end{bmatrix}$, $g_3 = \begin{bmatrix}0\\1\end{bmatrix}$ *Each of these vectors represents a direction in the plane.*

The zonotope we are constructing is centered at the origin, meaning that $c = \begin{bmatrix}0\\0\end{bmatrix}$. *To construct this zonotope, we sequentially add each vector to the set of points already drawn. Let's first imagine drawing a segment using vector g_1. Then, we take this segment and, from each of its points, we add vector g_2 to create a new shape. Finally, we repeat this process with vector g_3. At each step, we progressively expand the shape by adding new segments, always starting from the center. The result is a zonotope, which represents the convex hull of the points generated by this addition process. The illustration in the figure 1 visually demonstrates how this construction evolves as we add generators to the zonotope. Each new vector contributes to extending the shape's boundaries in different directions, resulting in a more complex geometric figure. The final zonotope is a centrally symmetric, convex polytope whose sides are parallel to the generating vectors. The number of faces and the overall complexity of the zonotope increase with each added generator, thus demonstrating the flexibility and utility of zonotopes in representing multidimensional spaces in a compact and structured manner.*

Zonotopes have favorable properties as they can be represented compactly, and they are closed under the Minkowski sum and linear mapping, For clarity, we will denote the real interval $[-1, 1]$ as $\mathcal{I}$.

Composition of Zonotopes.

Let Z_1 and Z_2 be two zonotopes defined by their centers c_1 and c_2, as well as their generator matrices $G_1 = \{g_{1,1}, g_{1,2}, \ldots, g_{1,d_1}\}$ and $G_2 = \{g_{2,1}, g_{2,2}, \ldots, g_{2,d_2}\}$, respectively. Then, the Minkowski sum of these two zonotopes is itself a zonotope and can be expressed as:

$$c_1 + G_1\,\mathcal{I}^{d_1} + c_2 + G_2\,\mathcal{I}^{d_2} = c + [G_1\ G_2]\,\mathcal{I}^{d}, \tag{5}$$

where $c = c_1 + c_2$, $d = d_1 + d_2$ and $[G_1\ G_2]$ represents the concatenation of the generators matrix of the two zonotopes. This illustrates that a zonotope is closed under the Minkowski sum.

Linear Multiplication of Zonotopes.

Let $L \in \mathbb{R}^{n\times m}$ be a matrix and $G = \{g_1, g_2, \ldots, g_d\} \subseteq \mathbb{R}^m$. Then, the linear multiplication of L by $G\,\mathcal{I}^d$ is also a zonotope and is expressed as:

$$L * G\,\mathcal{I}^d = (L * G)\,\mathcal{I}^d, \tag{6}$$

Symmetry of Generator Vectors.

Let Z be a zonotope defined by its generators $\{g_1, g_2, \ldots, g_d\}$. The zonotope remains the same if a generator vector g_i is replaced by $-g_i$, that is:

$$Z = c + \sum_{i=1}^{m} \mathcal{I} g_i \text{ is equivalent to } Z = c + \sum_{i=1}^{m} (-\mathcal{I}) g_i.$$

The central symmetry of a zonotope means that if a generator g_i is replaced by $-g_i$, the zonotope remains the same because every point of the zonotope can still be expressed as a linear combination of the modified generators.

Invariance under Permutation of Generator Vectors.

Let Z be a zonotope defined by its generators $\{g_1, g_2, \ldots, g_d\}$. Permuting the columns of the generator matrix does not change the zonotope, that is:

$$Z = [g_1\ g_2\ \ldots\ g_d]\,\mathcal{I}^d \text{ is equivalent to } Z = [g_2\ g_1\ \ldots\ g_d]\,\mathcal{I}^d.$$

Zonotopes are defined by the set of their generators and are independent of the order in which these generators are listed. Permuting the columns of the generator matrix does not affect the zonotope because it does not change the set of possible linear combinations of the generators.

▶ **Definition 2** (Hyperplanes). *A hyperplane H is the set of points $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that satisfy the equation:*

$$H = \{\mathbf{x} : \mathbf{a}^{\mathbf{T}} \cdot \mathbf{x} = \mathbf{b}\} \tag{7}$$

where:

- *$\mathbf{a} = (a_1, a_2, \ldots, a_n)$ is the normal vector, perpendicular to the hyperplane, and $\mathbf{a}^{\mathbf{T}}$ is its transpose.*
- *b is a scalar that determines the position of the hyperplane relative to the origin.*

In an n-dimensional space, a hyperplane is a subspace of dimension $n-1$. It generalizes the concepts of points, lines, and planes in higher dimensions.

▶ **Theorem 3** (Intersection of Convex Sets). *Let A and B be convex sets in a vector space. Then, their intersection $A \cap B$ is also a convex set.*

▶ **Theorem 4** (Hahn-Banach Separation Theorem). *Let C be a convex set in a finite-dimensional vector space. For any point x such that $x \notin C$, there exists a hyperplane that strictly separates x from C.*

3 Hyperplanes based zonotopic Contractor

The main advantage of contractors [5], [2], [11] is their ability to reduce the size of a box by narrowing the interval to retain only the desired set of data. The key idea is to refine the box while satisfying two important conditions:

1. The reduced box $[z]_0$ is always a subset of the original box $[z]$, meaning that no unnecessary data is included.
2. The intersection of the reduced box with the solution set S remains unchanged, ensuring that no potential solutions are lost.

In this work, we adopt a different approach aimed at reducing conservatism while preserving the accuracy of all potential solutions. Specifically, we leverage the use of zonotopes, which offer a more computationally efficient and less conservative alternative to intervals. By utilizing zonotopes, we are able to achieve a more precise representation of uncertainty, balancing computational manageability with improved solution integrity.

By definition, a zonotope is a convex set, and according to the theorem on the intersection of convex sets (Theorem 3) the intersection of two convex sets is also convex. However, this intersection does not necessarily result in a zonotope, which presents a challenge for efficient computation. Thus, a crucial step in our method is to approximate this convex intersection with a zonotope, simplifying subsequent calculations and manipulations.

To achieve this, hyperplanes are employed to enclose the intersection and construct a zonotopic approximation. The justification for using hyperplanes stems from the Hahn-Banach theorem (Theorem 4), which guarantees that two convex sets can always be separated by a hyperplane. This allows us to enclose any convex intersection with hyperplanes, creating a zonotopic envelope around the intersection. The process is carried out in three key steps:

1. **Selection of Non-Collinear Hyperplanes:** In this first step, the objective is to select two non-collinear hyperplanes, denoted as H_1 and H_2, which means that they are neither parallel nor aligned. Mathematically, this can be expressed by ensuring that the normal vectors a_1 and a_2 of H_1 and H_2 respectively are linearly independent, i.e., the rank of the matrix formed by a_1 and a_2 is equal to n:

$$\text{rank}([a_1, a_2]) = n \tag{8}$$

where $[a_1, a_2]$ is the concatenation of the normal vectors a_1 and a_2, and n is the dimension of the space in which the intersection of the zonotopes is defined.
These hyperplanes are specifically chosen to pass through facets of the intersection of two zonotopes, $Z_1 \cap Z_2$. Let F_i denote a facet of the convex hull of the intersection, which is a $(n-1)$-dimensional face. Each hyperplane H_i (for $i = 1, 2$) is defined such that it satisfies:

$$H_i : a_i^T x = b_i \quad \forall x \in F_i \tag{9}$$

2. **Construction of Parallel Hyperplanes:** Once the non-collinear hyperplanes H_i are defined, the next step involves constructing parallel hyperplanes to these. Each hyperplane is adjusted in space to be at a sufficient distance to encompass the entire convex intersection. This distance is determined by identifying the farthest vertex $\mathbf{v} \in V$ from the hyperplane H_i, where V is the set of vertices of the convex hull of the intersection. The distance from a vertex $\mathbf{v}$ to the hyperplane H_i is given by the formula:

$$d(\mathbf{v}, H_i) = \frac{|a_i^T \mathbf{v} - b_i|}{\|a_i\|} \tag{10}$$

The vertex $\mathbf{v}_{\max}$ corresponding to the maximum distance is selected:

$$\mathbf{v}_{\max} = \arg\max_{\mathbf{v} \in V} d(\mathbf{v}, H_i) \tag{11}$$

Once $\mathbf{v}_{\max}$ is identified, a parallel hyperplane H_i^{parallel} is constructed to pass through this vertex. The equation of the parallel hyperplane is defined as:

$$H_i^{\text{parallel}} : a_i^T \mathbf{x} = b_i^{\text{parallel}} \tag{12}$$

where $b_i^{\text{parallel}} = a_i^T \mathbf{v}_{\max}$

By selecting hyperplanes H_i that pass through all points x on the facet F_i, we minimize conservatism in the zonotope approximation of the intersection. Additionally, by taking the farthest vertex for the parallel hyperplane, we guarantee that the complete intersection is enclosed without loss.

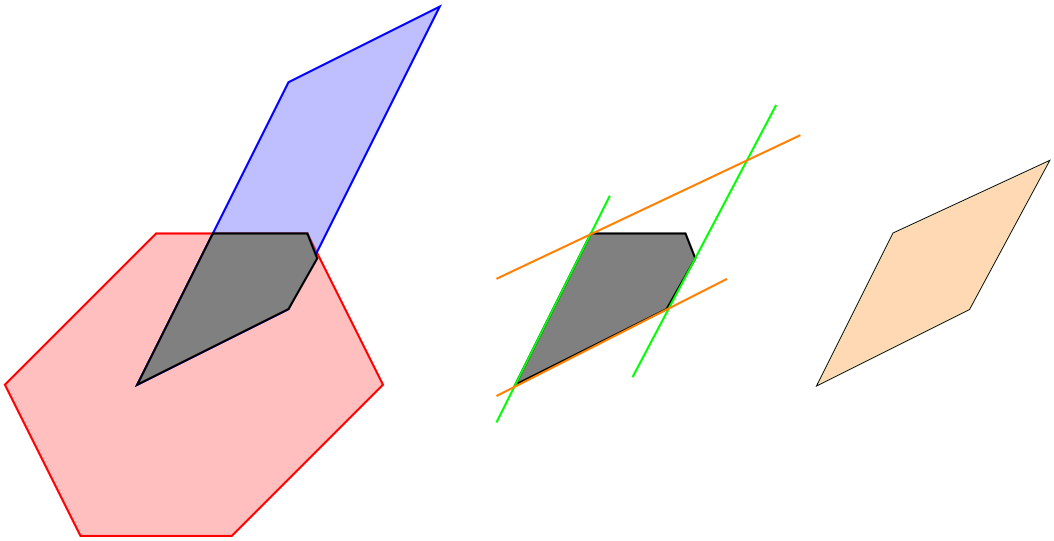


Figure 2 Illustration of the Zonotopic Contraction Method Using Hyperplanes.

This figure illustrates the key steps of the proposed zonotopic contraction method. The initial zonotopes are represented by the red and blue shapes, each generated by different sets of vectors. The intersection of these two zonotopes forms a complex convex region (in gray). To approximate this intersection while reducing conservatism, we use hyperplanes (depicted in green and orange) that enclose the intersection and define a new zonotopic boundary. The resulting zonotope, shown in orange, effectively captures the intersection of the original zonotopes, ensuring that the approximation remains both accurate and computationally manageable. This process is iteratively refined to minimize the volume of the final zonotope while preserving the integrity of all potential solutions within the intersection.

3. **Volume Comparison:** Once the hyperplanes are defined and placed, we proceed with an iterative volume comparison. The goal is to adjust the positions of the hyperplanes by making them pass through other edges of the intersection, if any remain, and to check all possibilities in order to find the optimal combination. This approach allows us to minimize the volume of the final zonotope while ensuring that the intersection is properly enclosed.

4 Model and simulation results

To test the proposed contractor, the commonly known "Linear Bicycle Model" [10] is used.

The considered continuous time model and the linearization assumptions are presented as follows:

1. **State Variables:**
   - $\beta(t)$: the sideslip angle, representing the difference between the direction of the vehicle's movement and its orientation.
   - $\psi(t)$: the yaw angle, corresponding to the vehicle's rotation around its vertical axis.

2. **Forces:**
   - $F_{tyf}(t)$: lateral forces on the front tires.
   - $F_{tyr}(t)$: lateral forces on the rear tires.
   - $F_{txf}(t)$: longitudinal forces on the front tires.
   - $\Delta F_{txr}(t)$: differential braking force on the rear tires, calculated from the braking torques T_{br}.
3. **Linearization Assumptions:**
   - The sideslip angles $|\beta|$ are small, less than 7 degrees.
   - The longitudinal slip ratio is small, less than 0.1.
   - The steering angles δ are small, allowing the approximation $\cos(\delta) \approx 1$.

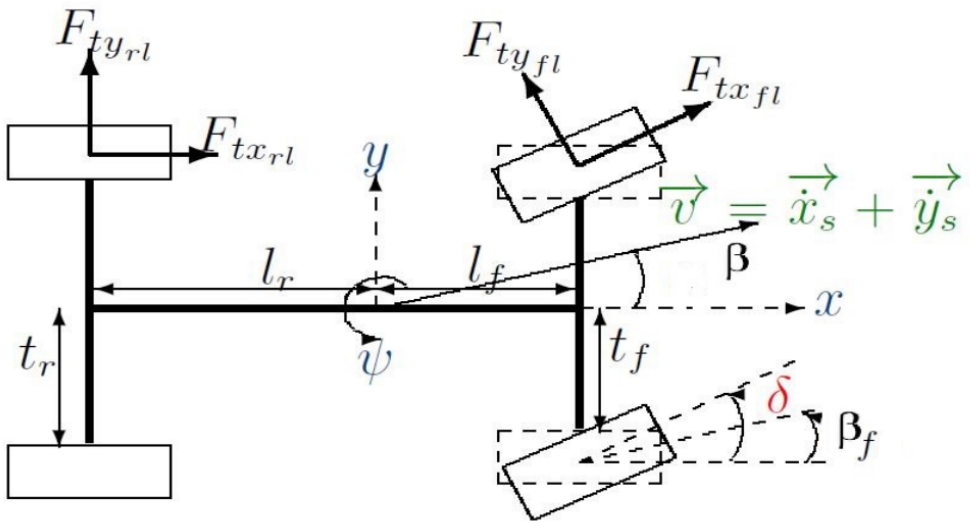


Figure 3 Representation of the bicycle model illustrating the car's lateral behavior [10].

The linearized model yields the following equations for lateral tire forces:

$$F_{tyf}(t) = C_f \beta_f(t) \tag{13}$$

$$F_{tyr}(t) = C_r \beta_r(t) \tag{14}$$

where $\beta_f(t)$ and $\beta_r(t)$ are the front and rear sideslip angles, given by:

$$\beta_f(t) = \delta(t) - \beta(t) - \frac{l_f \dot{\psi}(t)}{v}, \tag{15}$$

$$\beta_r(t) = \beta(t) + \frac{l_r \dot{\psi}(t)}{v}. \tag{16}$$

This leads to the following state-space representation:

$$\begin{bmatrix} \dot{\beta}(t) \\ \dot{\psi}(t) \end{bmatrix} = \begin{bmatrix} -\frac{C_f + C_r}{mv} & 1 + \mu - \frac{l_r C_r + l_f C_f}{mv^2} \\ -\frac{l_r C_r + l_f C_f}{I_z} & -\frac{l_f^2 C_f + l_r^2 C_r}{I_z v} \end{bmatrix} \begin{bmatrix} \beta(t) \\ \psi(t) \end{bmatrix} + \begin{bmatrix} \frac{C_f}{m} & 0 & 0 & 0 \\ \frac{l_f C_f}{I_z} & \frac{1}{I_z} & \frac{S_r R t_r}{2I_z} & -\frac{S_r R t_r}{2I_z} \end{bmatrix} \begin{bmatrix} \delta \\ M_{dz} \\ T_{brl} \\ T_{brr} \end{bmatrix} \tag{17}$$

The parameters used in the previous formulas are defined as follows:

- m: Vehicle mass, which is 1535 kg.
- I_z: Vehicle yaw inertia, equal to 2149 kg $\cdot$ m^2.
- C_f: Lateral stiffness of the front tires, given as 20000 N/degree.
- C_r: Lateral stiffness of the rear tires, also 20000 N/degree.
- S_r: Longitudinal stiffness of the rear tires, valued at 12720 N.
- l_f: Distance from the center of gravity (COG) to the front axle, which is 1.4 meters.
- l_r: Distance from the center of gravity to the rear axle, set at 1 meter.
- t_r: Length of the rear axle, specified as 1.4 meters.
- R: Tire radius, with a value of 0.3 meters.
- μ: Tire-to-road friction coefficient, ranging from 0.4 to 1.
- v: Vehicle speed, ranging from 50 to 130 km/h.
- δ: Steering angle.
- M_{dz}: Yaw moment disturbance.
- T_{brj}: Braking torques at the left and right rear tires (where $j = \{l, r\}$).

These parameters are crucial for modeling the vehicle's lateral dynamics. These values were obtained through an identification process on the Renault Mégane Coupé (see [4]).

Note that the bicycle model has been discretized to be used for the simulation. The filter undergoes the following steps:

1. Simulation Step :
   The equations for the actual system state and measured output are given by:

$$\begin{cases} x_{\text{real}}(k) = Ax_{\text{real}}(k-1) + Bu(k) \\ [y_{\text{measured}}]_z = Cx_{\text{real}}(k) + [v]_z \end{cases} \tag{18}$$

   where:
   - $x_{\text{real}}(k)$ is the time derivative of the actual state vector at time step k.
   - A and B are system matrices.
   - $u(k)$ represents the input vector at time step k.
   - C is the output matrix (identity matrix).
   - $[v]_z$ denotes the bounded noise affecting the measurements.

   Note the that the real state and the measurement are transformed into zonotopes to cope with the filter requirements.
2. Propagation Step:
   The equations for predicting and propagationg the next state and output are:

$$\begin{cases} [\hat{x}(k)]_z = A[\hat{x}(k-1)]_z + Bu(k) + [w]_z \\ [\hat{y}(k)]_z = C[\hat{x}(k)]_z + [v]_z \end{cases} \tag{19}$$

   where:
   - $[\hat{x}(k)]_z$ is the predicted zonotope state at time step k.
   - $[\hat{y}(k)]_z$ is the predicted zonotope measure at time k.
   - $[w]_z$ denotes the bounded noise affecting the system's dynamics.
3. Correction Step : The correction of the estimated state $\hat{x}(k)$ based on the measured output is performed using:

$$\begin{cases} intersect = [\hat{y}(k)]_z \cap [y_{\text{measured}}]_z \\ \hat{x}^r(k) = [intersect]_z \end{cases} \tag{20}$$

   where $[intersect]_z$ is an approximate zonotope representation that is close to the true intersection but simplifies the computation.

Here, $\hat{y}(k)$ is directly equivalent to $\hat{x}(k)$ due to the identity matrix C. Thus, the correction step adjusts the estimated state $\hat{x}^r(k)$ to better match the measured output y_{measured}, considering any discrepancies or noise in the measurements.

The system is represented by the following state-space matrices:

$$A = \begin{bmatrix} -\frac{C_f+C_r}{mv} & 1-\frac{l_rC_r+l_fC_f}{mv^2} \\ -\frac{l_rC_r+l_fC_f}{I_z} & -\frac{l_f^2C_f+l_r^2C_r}{I_zv} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{C_f}{mv} & 0 & 0 & 0 \\ \frac{l_fC_f}{I_z} & \frac{1000}{I_z} & \frac{Rt_r}{2I_z} & \frac{-Rt_r}{2I_z} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The initial conditions are set as follows:

- The initial state vector x_0 is $[0, 0]$.
- The initial zonotope $[x_0]_z$ is initialized with a center of $[0, 0]$ and generators $[0, 0.5; 0.5, 0]$.

Input values u_k, which include steering angles, yaw moment disturbances, and braking torques, are loaded for simulation.

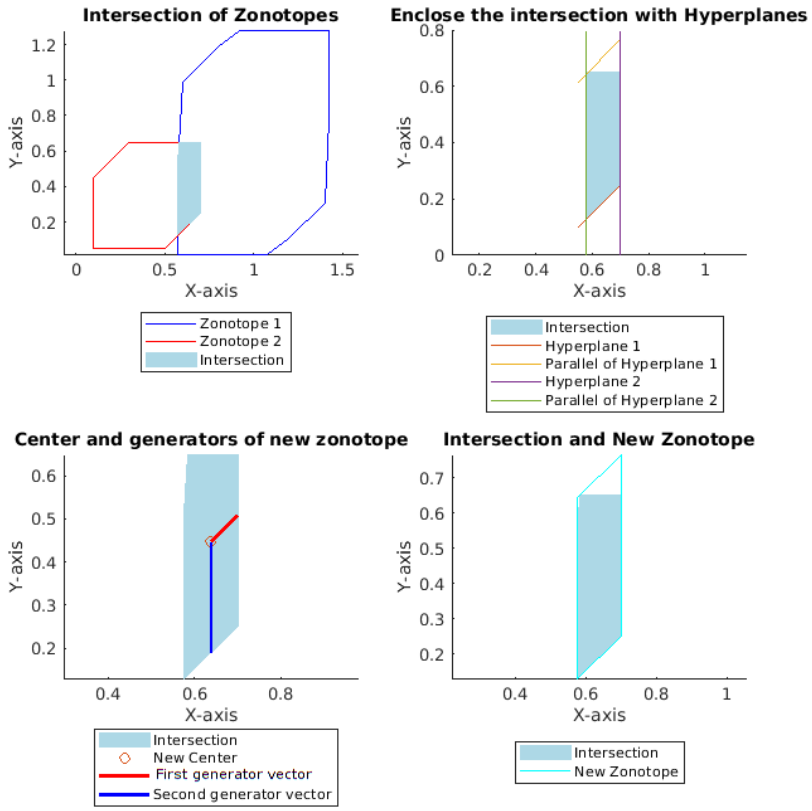


Figure 4 Example execution of the zonotope contractor method with $k = 30$ and an error rate of 0.1131.

The error rate is defined as the relative difference between the volume of the intersection of the predicted and actual zonotopes and the volume of the new zonotope. It is calculated as follows:

$$\text{error_rate} = 1 - \frac{\text{volume(intersect)}}{\text{volume(new_zonotope)}} \tag{21}$$

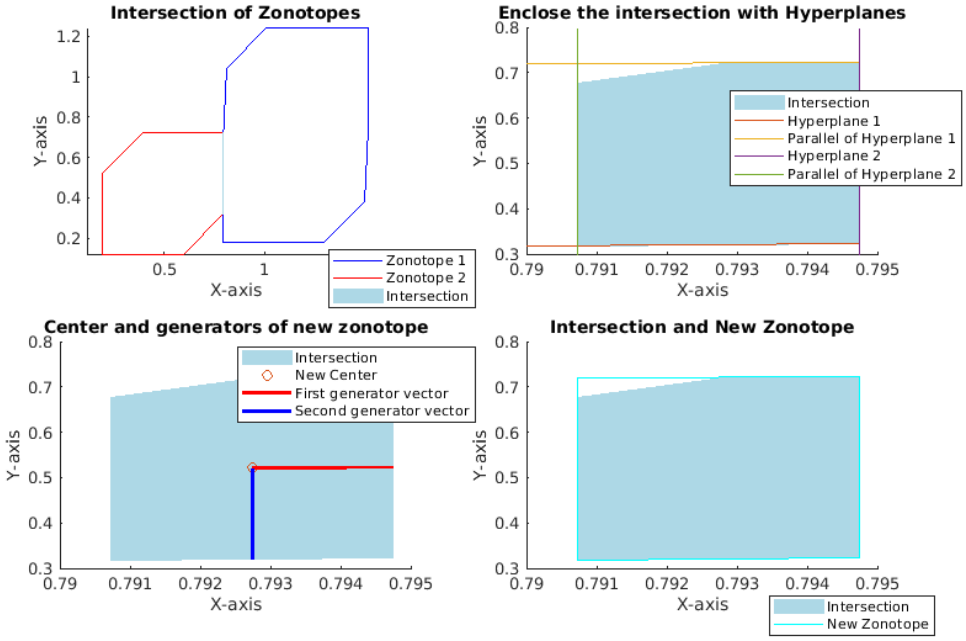


Figure 5 Example execution of the zonotope contractor method with $k = 600$ and an error rate of 0.03035.

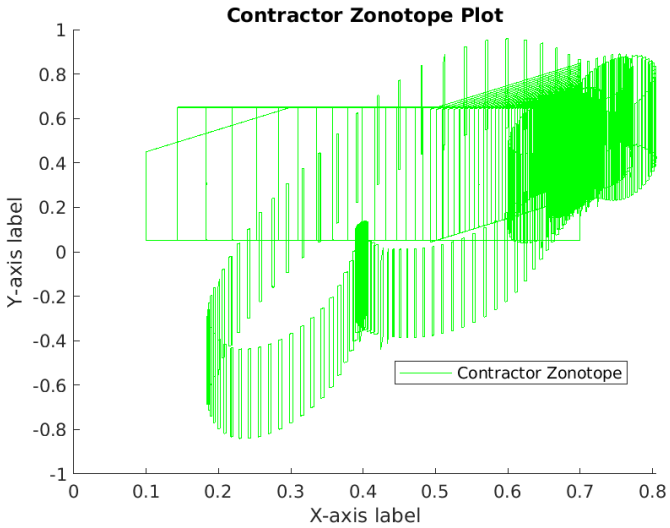


Figure 6 Contractor Zonotope over Iterations. This plot shows the evolution of the contractor zonotope as it becomes tighter with each iteration, illustrating how the method increasingly refines the estimated state.

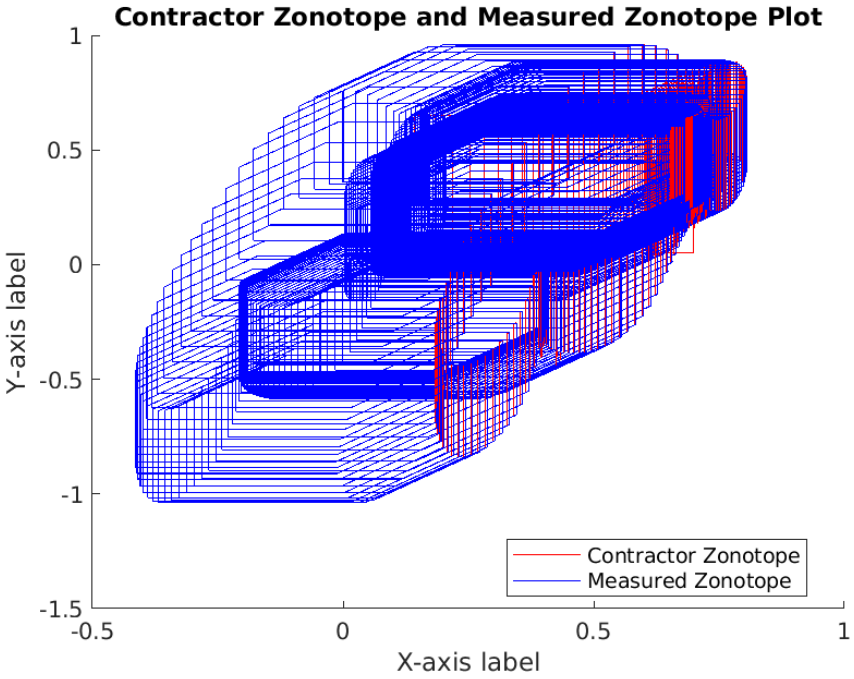


Figure 7 Comparison of the contractor zonotope estimates (in red) with the measured zonotopes (in blue) over $N = 864$ iterations. The contractor zonotope progressively tightens around the true state, demonstrating improved accuracy in state estimation as iterations advance.

The results demonstrate that the hyperplane based zonotopic contractor provides increasingly tighter estimates of the intersection as the iteration progresses from $k = 1$ to $k = N = 864$. This tightening is reflected in the improved accuracy of the state estimation, as the contractor becomes more refined with each iteration, effectively narrowing the uncertainty in the intersection with the measured zonotope.

Moreover, the error rate remains relatively low throughout the iterations. This low error rate indicates that the proposed contractor effectively approximates the true intersection, capturing the system's behavior with high fidelity. The tight bounding of the intersection suggests that it is a reliable method for state estimation in the presence of uncertainties and noise, making it a robust tool for systems modeled using zonotopes

5 Conclusion and future works

This work presents a novel method for zonotopic contraction using hyperplanes, with the goal of reducing conservatism while preserving the zonotope's structural integrity. The key advantage of this approach is its ability to maintain a zonotopic structure that closely approximates the convex intersection of the original zonotopes. By leveraging the properties of hyperplanes, we accurately approximated this intersection, ensuring that the resulting zonotope remains a manageable convex set. Through the use of non-collinear and parallel hyperplanes, we systematically refined the zonotope boundaries, minimizing volume while retaining the critical characteristics of the intersection. Although we cannot prove the optimality of this approach in all scenarios, we provide strong guarantees that it reduces conservatism and captures all potential solutions within the intersection. This method's

increased precision is especially relevant in safety-critical systems such as autonomous driving and robotics. For example, in autonomous vehicle navigation, the accurate capture of the intersection of state spaces represented by zonotopes leads to better trajectory planning and improved obstacle avoidance, thereby reducing collision risks and ensuring safer navigation in uncertain environments. Furthermore, in control and fault detection systems, this approach enhances state estimation accuracy and fault diagnosis. In fields like industrial robotics and aerospace, precise zonotopic contraction improves uncertainty management, optimizing system performance and reducing failure risks.

Another point concerns the computational complexity, particularly in Step 3 (Volume Comparison). Currently, the recursive volume comparison remains manageable with the use of two hyperplanes. However, as we extend the method to incorporate more hyperplanes in higher-dimensional spaces, we anticipate an increase in computational complexity. This potential increase has been taken into account, and future works will focus on optimizing the algorithm to mitigate this impact. Our current approach is designed to retain the inherent simplicity of zonotopes while exploring more efficient strategies for volume minimization. At this stage, the computational overhead remains low with the two-hyperplane scenario. Nevertheless, as we expand to higher dimensions and incorporate additional hyperplanes, we are actively pursuing optimization strategies for the recursive process. Our goal is to strike a balance between computational efficiency and accuracy, enabling the method to scale effectively with increasing dimensionality.

References

1 Matthias Althoff. On computing the minkowski difference of zonotopes. *arXiv preprint arXiv:1512.02794*, 2015. arXiv:1512.02794.

2 Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009. doi:10.1016/J.ARTINT.2009.03.002.

3 Felix L. Chernousko. Ellipsoidal state estimation for dynamical systems. *Nonlinear Analysis: Theory, Methods & Applications*, 63(5-7):872–879, 2005.

4 Soheib Fergani. *Robust multivariable control for vehicle dynamics*. PhD thesis, Université de Grenoble, 2014.

5 Luc Jaulin, Michel Kieffer, Olivier Didrit, Eric Walter, Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Interval analysis*. Springer, 2001.

6 Rudolf Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering*, 82:35–45, 1960.

7 Wolfgang Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998. doi:10.1007/BF02684450.

8 Wolfgang Kühn. Zonotope dynamics in numerical quality control. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 125–134. Springer, 1998. doi:10.1007/978-3-662-03567-2_10.

9 Jinsun Liu, Yifei Simon Shao, Lucas Lymburner, Hansen Qin, Vishrut Kaushik, Lena Trang, Ruiyang Wang, Vladimir Ivanovic, H. Eric Tseng, and Ram Vasudevan. Refine: Reachability-based trajectory design using robust feedback linearization and zonotopes. *IEEE Transactions on Robotics*, 40:2060–2080, 2024. doi:10.1109/TRO.2024.3366819.

10 Quoc Hung Lu. *Filtrage à incertitudes stochastiques et bornées: application au diagnostic actif en automobile*. PhD thesis, Université Paul Sabatier-Toulouse III, 2022.

11 Moussa Maiga, Nacim Ramdani, Louise Travé-Massuyès, and Christophe Combastel. A comprehensive method for reachability analysis of uncertain nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 61(9):2341–2356, 2015. doi:10.1109/TAC.2015.2491740.

12 Tuan Anh Tran. *Cadre unifié pour la modélisation des incertitudes statistiques et bornées: application à la détection et isolation de défauts dans les systèmes dynamiques incertains par estimation*. PhD thesis, Université Paul Sabatier-Toulouse III, 2017.

13 Bo Zhou, Jianda Han, and Guangjun Liu. A ud factorization-based nonlinear adaptive set-membership filter for ellipsoidal estimation. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 18(16):1513–1531, 2008.

14 Günter M Ziegler. Graduate texts in mathematics, 1995.

On a Method to Measure Supervised Multiclass Model's Interpretability: Application to Degradation Diagnosis

Charles-Maxime Gauriat
LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
Robert BOSCH (SAS), Paris, France

Yannick Pencolé
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Pauline Ribot
LAAS-CNRS, Université de Toulouse, UPS, Toulouse, France

Gregory Brouillet
Robert BOSCH (SAS), Paris, France

Abstract

In an industrial maintenance context, degradation diagnosis is the problem of determining the current level of degradation of operating machines based on measurements. With the emergence of Machine Learning techniques, such a problem can now be solved by training a degradation model offline and by using it online. While such models are more and more accurate and performant, they are often black-box and their decisions are therefore not interpretable for human maintenance operators. On the contrary, interpretable ML models are able to provide explanations for the model's decisions and consequently improves the confidence of the human operator about the maintenance decision based on these models. This paper proposes a new method to quantitatively measure the interpretability of such models that is agnostic (no assumption about the class of models) and that is applied on degradation models. The proposed method requires that the decision maker sets up some high level parameters in order to measure the interpretability of the models and then can decide whether the obtained models are satisfactory or not. The method is formally defined and is fully illustrated on a decision tree degradation model and a model trained with a recent neural network architecture called Multiclass Neural Additive Model.

2012 ACM Subject Classification Computing methodologies → Machine learning

Keywords and phrases XAI, Interpretability, multiclass supervised learning, degradation diagnosis

Digital Object Identifier 10.4230/OASIcs.DX.2024.27

Category Short Paper

1 Introduction

Condition monitoring plays an inevitable role in the safety of any industrial system, especially when it comes to the sensitive parts of machines, like the bearings in rotating machinery, which are prone to faults. Fault diagnosis of rotating machinery is a technique of fault detection, isolation and identification, which can be used as an assistance for system maintenance. Essentially, fault diagnosis can be regarded here as a pattern recognition problem that aims at determining the current degradation state of the rotating machinery based on the available set of measurements and its possible future trend (prognostics). As a powerful pattern recognition tool, Artificial Intelligence (AI) and especially Machine Learning (ML) has attracted great attention from many researchers and shows promise in rotating machinery fault recognition applications [7, 2]. While the main reason for using ML techniques is usually the models' performance (accuracy score, computational speed), the question about the


© Charles-Maxime Gauriat, Yannick Pencolé, Pauline Ribot, and Gregory Brouillet;
licensed under Creative Commons License CC-BY 4.0



ability of humans to understand them is of great importance. The interpretability of models is essential as soon as these models are effectively used in practice to solve problems and provide decisions for humans and/or for their businesses [23]. In our industrial context, such models are used to solve maintenance decision problems. Maintenance consists in optimally deciding when to replace a component in a system (like a machine tool) so that the system is always operating properly and manufacturing waste is prevented. To get such a maintenance strategy, the objective is to add relevant sensors in the system, to acquire time series at operating time and use a degradation model to check the current health of every component based on these time series. Amongst these ML methods, Neural Network approaches are more and more used as they can handle large and complex computations to produce efficient models. However, even if already proposed ML methods greatly improve the degradation diagnosis of such equipments, degradation models obtained by deep learning techniques are known to be black-boxes, meaning that they cannot be open to understand their decisions as interpretable. Some previous works have reached a certain level of explainability like [17] but these results do not give the full insight about how interpretable the effective model's choices are. However, it is important for a human operator to understand how an algorithmic model determines a maintenance decision with respect to human-interpretable physical laws and quantities: *how and why* such a model plans the decision. In fact, by providing the physical reason why the model decides about a degradation level, model's interpretability not only provides a diagnosis of the equipment but also improves the confidence needed by a human operator towards trained models [18].

In this paper, we address the problem of how to effectively and quantitatively measure the interpretability of a pretrained model that results from a Multiclass Supervised Learning problem and apply it to pretrained degradation models. The purpose of this multi-factor interpretability score is to let business skateholders, like maintenance operators, decide whether the obtained model is sufficiently interpretable for their own use. The proposed method is agnostic in the sense that it does not rely on a specific ML technique. We actually aim at applying it to different ML techniques that are currently used to solve the maintenance problem (i.e. Decision Tree (DTs), Multi-Layer Perceptrons (MLPs), Neural Additive Model for multi-class supervised learning (MNAMs) [9]). We also propose that the interpretability measure is parameterized, and the initialization of these parameters is the responsability of the human decision maker.

The paper is organized as follows. Section 2 first discusses the notion of interpretability with respect to the notion of local/global explainability in models and details related work about how to score interpretability. Section 3 describes a first degradation model as a simple decision tree model that will be used throughout this paper as an illustration of the concepts that we introduce to measure interpretability. Section 4 formally defines the way to score interpretability. Finally, Section 5 discusses the way to apply the interpretability scoring framework to a second degradation model that has been trained with our own recent neural network architecture: Multiclass Neural Additive Models (MNAMs) [9].

2 About interpretable models

2.1 Concepts and definitions

Lately, the Explainable Artificial Intelligence (XAI) community has been using various terms referring to the comprehension of machine learning models: interpretability, explainability, intelligibility or even comprehensibility [20]. As the vision behind these concepts seems to be fuzzy and does not refer to a monolithic concept so far [14], we have decided in this paper to propose the following definitions for explainability and interpretability.

Explainability of a machine learning model is based on the ability to obtain rules that highlight the relations between attributes and predictions. These rules are conditional functions like: "IF A $>$ 1 and B $<$ 4 THEN PREDICTION P" with two features A and B in this example. Based on these rules, two types of explainability can be defined: local explainability and global explainability.

Local explainability means the ability of identifying, for a given prediction, the rule that shows the relation between the feature and that specific prediction. The rule explains why a particular prediction was made based on the features of the instance in question. Local explainability is inherent in Decision Trees (DT) because the rule corresponding to a specific prediction can be extracted from the leaf node where the prediction concludes. For other methods such as MLP, it can be difficult to extract such a rule because of the complex interactions between features and the non-linearity of the model. To address this challenge, various post-prediction processing methods have been developed. For instance, these include extracting rules from the weights and activations of a MLP [8]. Additionally, widely-used methods, such as Local Interpretable Model-agnostic Explanations (LIME) [22] and SHapley Additive exPlanation (SHAP) [17], can be used to approximate the rules linking features to predictions. However, all these methods only provide approximations of the rules and, therefore, do not always accurately reflect the model's decision-making process. Other methods have focused on the extraction of symbolic rules [12], [25], with the most efficient recent framework being Deep Red [28] which extract approximate IF-THEN-ELSE rules from neural networks.

Global explainability is a stronger concept than local explainability, achieved by extracting the entire set of rules used by a model. This allows for a general understanding of how the model operates and how it will make decisions for different instances. Whatever the predictions the model will perform in the future, they can be explained by these feature-based rules. Global explainability ensures the algorithmic transparency of the models because of this set of rules; it becomes possible to determine the classification of any instance without explicitly running it through the model. DT are inherently globally explainable. A DT rule is defined by a branch of the tree and the thresholds defined at each node along the branch. All rules can be displayed, allowing a human to manually predict outcomes by following these rules. In contrast, Multi-Layer Perceptrons (MLP) do not provide precise model rules because a fully connected network is highly complex. Therefore, MLPs are not globally explainable.

Interpretability is the model's ability to be understood by humans. This is achieved by keeping the explanation as minimalist as possible and is therefore a more quantitative property as opposed to global explainability. The fewer the number of rules and the fewer the number of features used within to make a prediction, the higher the interpretability will be. According to our definitions, interpretability implies global explainability. Interpretability is bringing up together concepts as simulatability, decomposability from [14] and comprehensibility from [20] which are about maximizing the human comprehension using the minimum set of rules. Any type of model may lack of interpretability. For instance, a DT may have a high number of rules or be excessively deep. As stated above, globally explainable models rely on the fact that it is possible to associate their decision to a set of rules. However, rule extraction methods often prioritize the fidelity of the extracted rules over their quality [3, 27] which leads to a lack of interpretability. In [28], for instance, the quality of the rules is evaluated in terms of their accuracy and fidelity to the original model, while their comprehensibility is secondary and measured only by their length and simplicity.

This paper addresses the problem of scoring interpretability of a globally explainable model whatever methods have been used to extract the model's rules. We aim at measuring how interpretable the model's decisions are.[1]

2.2 Metrics for interpretability: related work

A survey in [6] offers a comprehensive review of causal interpretable models, discussing methods, and evaluation metrics for interpretability. [26] proposes to measure a model's interpretability using a decision tree-based model, but the selection criteria for learning are based on human perception and lack formal precision. Other works deal with optimised rule extraction under various constraints in order to obtain an interpretable result [5, 15]. The interpretability of decision trees was enhanced in [24] by reducing the number of distinct attributes required to explain classifications. The SER-DT algorithm, minimizes the explanation size (i.e., the number of attributes used) while maintaining high accuracy and interpretability. A trade-off between tree depth and explanation size is introduced, demonstrating that both can be optimized simultaneously. The concept of explanation size is presented as a new metric for assessing interpretability. In [4] both length and coverage are metrics for assessing rule interpretability. Length refers to the number of conditions in a rule; shorter rules are more interpretable due to their simplicity. Coverage measures how many data points a rule applies to; broader coverage indicates greater generality, making the rule easier to understand and reducing the need for many low-coverage rules. A quantifiable interpretability score is also proposed in [19] to evaluate and compare rule-based and tree-based algorithms. This score is obtained from a weighted sum of three metrics: predictivity, stability and simplicity. Predictivity measures the accuracy of the model's predictions, ensuring trust in its decision-making capabilities. Stability evaluates how robust the model's rules are to changes in the input data, ensuring that small perturbations do not drastically alter the generated rules. Simplicity is defined by the length and number of rules. The shorter and fewer the rules, the more interpretable the model is considered to be.

In this paper, we propose a parameterized method to score the interpretability of any globally explainable model and to filter out non interpretable rules. This method combines into a unique score two metrics: namely the accuracy and the coverage of the rules.

3 Running Example

The way to score model's interpretability will be illustrated all along this paper with the following example that is a multi-class supervised learning problem. This example has been selected as it is based on real data, and it is simple.

3.1 Experimental setup

In Bosch-Rodez, we set up an experimental platform (full details in [10]) that records time series on a set of spindles (critical parts of machine tools under maintenance). Figure 1 illustrates this experimental platform. To initiate the experiment and get the training measured data, an available set of five spindles has been installed on the test-bed in a closed chamber that simulates the real operating conditions. Each spindle is connected with cooling oil inlet and outlet for cooling down the spindle (lubrication of the bearings). Then a VSA

[1] Note that this paper does not discuss the problem of model accuracy. A model can make wrong decisions that are perfectly interpretable.

005 sensor (the accelerometer) has been screwed on the front part of the spindle as it would be inside a machine tool using this type of spindle at operating time. This vibration sensor is thus located between the pair of bearings of the spindle. It measures the vertical acceleration just on the top of these bearings. The cable of the VSA sensor is connected to an IFM VSE 100 module that contains the software allowing to record the vibratory signals, a module that is used at operating time. The objective is to acquire data at 9K RPM so that frequency measurements are within the range of the available VSA sensor.

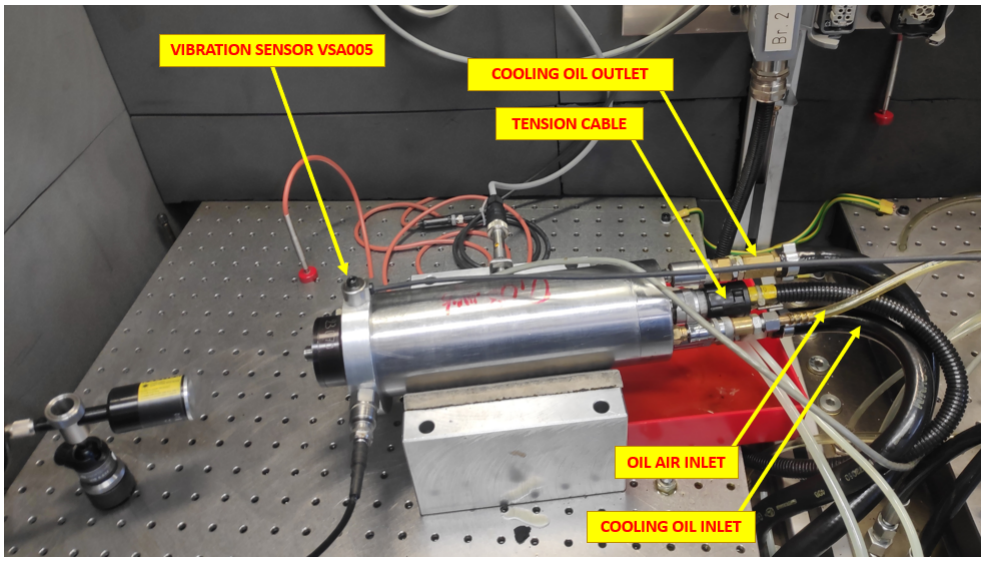


Figure 1 Experimental test-bed.

These time series consist of vibration data. The degradation state of each spindle that is measured is known by expertise and leads to five classes $\mathcal{C} = \{1, \ldots, 5\}$ that represents the degradation of a spindle: from Class 1 that characterises new spindles to Class 5 that characterises worn-out and failing spindles. In this context, the objective of degradation diagnosis is to be able to determine at operating time what is the effective class of the measured spindle. Especially if the operating spindle is in Class 4, its replacement would be necessary as it is about to fail.

3.2 Degradation model

Our objective is then to learn from the time series a degradation model M that is able to predict the degradation class $c \in \mathcal{C}$ of a spindle. To do so, available raw time series have been converted to spectrograms (short fast Fourier transforms) so that each time series is represented by a set of 409 frequency amplitudes (from 0Hz to 9985Hz with a resolution of 24.414Hz). We denote by X the available dataset, each individual $x \in X$ is then composed of 409 features. In the following, $fa(x, f)$ will denote the frequency amplitude at the frequence f in the individual x. Experts were able to label any individual x with an ageing class denoted $\ell(x) \in \mathcal{C}$.

Figure 2 presents the degradation model M that has been trained on these labeled data. It is a DT that can be used to assist the maintenance decision. For instance, if at operating time, a new individual x is available and if $fa(x, 3800) \leq 186.224$ and $fa(x, 175) > 277.893$, the model M predicts the spindle is currently in class 2 and no replacement is required yet. Each node is also associated with the following information: N_i is the number of individuals

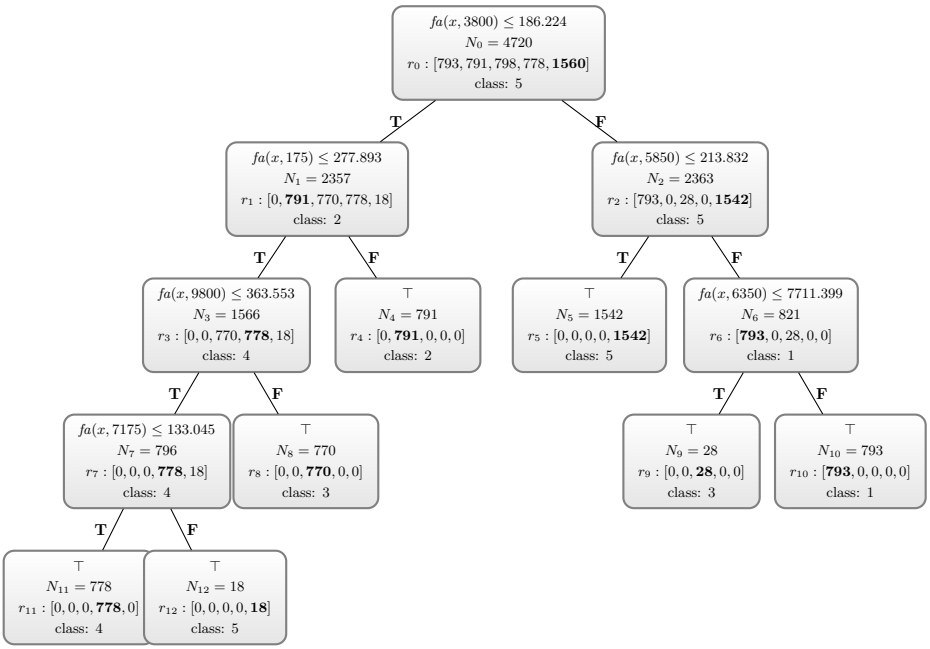


Figure 2 Degradation model of a spindle based on vibration frequencies.

from X that are covered by the node, and a vector of numbers that shows the distribution of these individuals with respect to their label. For instance the root node covers 4720 individuals (that is the size of X), 793 individuals are labeled 1, 791 are labeled 2, etc.

While this DT has been trained on real data, it must be however noticed that this model is not satisfactory yet in terms of performance and accuracy as the available dataset X is still limited and more recordings are required to enrich the dataset X. This model is however satisfactory for the sake of illustration of the way to measure its interpretability.[2]

4 Rule-based interpretability score

This section formally introduces a method to score the interpretability of a model that is globally explainable. For the sake of illustration, this method is applied to the DT that is detailed in Section 3. As detailed in Section 2, a model M is globally explainable if there exists a set of feature-based rules that can be used to provide an explanation for any prediction.

In the following, A denotes the set of available attributes of the problem. An individual is generally denoted $x = (x_1, \ldots, x_{|A|})$ where x_i is the value of x associated with the i^{th} attribute of A. The space of every possible individuals is denoted $\mathcal{X}$ and the available dataset for training the model is denoted X, obviously $X \subseteq \mathcal{X}$. Let B denote a subset of A, $\mathcal{X}_B$ denotes the projection of $\mathcal{X}$ on the attributes in B.

▶ **Definition 1** (Rule). *Let B be a subset of features A, a* rule *r over B on a class $c \in \mathcal{C}$ is a Boolean function*

$$r : \begin{array}{l} \mathcal{X}_B \to \{\mathbf{T}, \mathbf{F}\} \\ x_B \mapsto r(x_B). \end{array} \tag{1}$$

[2] Interpretability does imply accurracy and reciprocally.

The purpose of a rule r is to assert whether the prediction of an individual $x \in \mathcal{X}$ is c or not *according to the rule*. Such a rule can be used as an explanation of why the prediction of individual x given by the underlying model M is class c:

$$M \text{ predicts class } c \text{ for individual } x \textit{ because } r(x) \text{ is true.}$$

As opposed to a model, a rule r is usually defined over a subset of features B. This subset is called the *support* of rule r. In the following, $r(x)$ denotes the effective prediction $r(x_B)$ where B is the support of r.

In Figure 2, each node of the DT is associated with a rule. For instance, consider the node associated with rule r_3. This node is associated with class 4 as it is the class of the largest set of individuals from the training dataset X covered by this node (i.e. 778 individuals over 1566). Rule r_3 associated with this node informally states that any individual whose frequency amplitude at $3800Hz$ is smaller than 186.22 and frequency amplitude at $175Hz$ is smaller than 277.893 should be of class 4. The support of r_3 only consists of two attributes (frequency amplitude at $3800Hz$ and frequency amplitude at $175Hz$) over 409 attributes. Rule r_3 on class 4 is formally defined as follows:

$$r_3(x) \text{ iff } fa(x, 3800) \leq 186.224 \wedge fa(x, 175) \leq 277.893$$

Another example is rule r_5 on class 5 that is associated with a leaf node of the DT. The support of rule r_5 is the frequency amplitudes at $3800Hz$ and $5850Hz$, formally:

$$r_5(x) \text{ iff } fa(x, 3800) > 186.224 \wedge fa(x, 5850) \leq 213.832$$

Note that the root node of the tree is also associated with a rule r_0 on class 5. The support of r_0 is empty and $\forall x \in \mathcal{X}, r_0(x)$.

▶ **Definition 2** (Rule coverage). *Let r be a rule on a class c, the* coverage *of the rule r over a dataset X is:*

$$cvr(r) = \{x \in X, r(x)\}. \tag{2}$$

The correct coverage *of the rule r over a dataset X is:*

$$ccvr(r) = cvr(r) \cap \{x \in X, \ell(x) = c\}. \tag{3}$$

Two rules r_1 and r_2 are disjoint if:

$$cvr(r_1) \cap cvr(r_2) = \varnothing. \tag{4}$$

Back to Figure 2, the dataset X is composed of $N_0 = 4720$ individuals and rule r_0 covers all of them ($|cvr(r_0)| = N_0$). Generally speaking, for every rule r_i in Figure 2, $|cvr(r_i)| = N_i$. Any node of the tree displays the size of the associated set $ccvr(r)$ in bold: for example, $|ccvr(r_1)| = 791$ (among 2357 individuals covered by r_1, only 791 are labeled with class 2). Disjoint rules in a DT are rules that do not belong to the same branch. For instance, rules r_{12} and r_5 on class 5 are disjoint, while r_2 and r_5 are not (rule r_2 subsumes r_5).

▶ **Definition 3** ((τ, ε)-rule). *Let $\tau \in [0, 1]$, $\varepsilon \in [0, 1]$, a (τ, ε)-rule on a class c over a dataset X is rule r such that:*

$$\frac{|ccvr(r)|}{|\{x \in X, \ell(x) = c\}|} \geq \tau, \qquad \frac{|cvr(r) \setminus ccvr(r)|}{|cvr(r)|} \leq \varepsilon \tag{5}$$

Parameter τ is a *minimal coverage rate* that is selected by the decision maker before the extraction of a set of interpretable rules. A rule r with an effective coverage rate lower than τ is considered by the decision maker not enough significant as it covers too few individuals from the dataset. Rules with low coverage rate might in fact be due to overfitting and could not be considered as interpretable. Parameter ε is the *maximal error rate*. This error rate is also selected by the decision maker before the extraction of a set of interpretable rules. If a rule on a class c has a low error rate ε, it means that the explanation provided by the rule for class c is given with a high level of confidence $1-\varepsilon$. Individuals misclassified by the rule and included in the error rate can be considered anomalies or noise if this rate is low. This is valid within the context of the dataset used to train the model, as it reflects the model's perspective based on the data it has seen.

Here are a few examples based on Figure 2. Rule r_2 is on class 5, $|ccvr(r_2)|$ is 1542. The number of individuals in X labelled with class 5 is 1560 (see details in node r_0). So the coverage rate of r_2 is $(1542/1560) = 98.89\%$. Looking now at the error rate of r_2, it is given by $(2363-1542)/2363 = 34.74\%$. Therefore, by Definition 3, rule r_2 is a (τ,ε)-rule for any couple (τ,ε) such that $\tau \in [0, 0.9889]$ and $\varepsilon \in [0, 0.3474]$. While r_2 is covering most individuals of class 5 in X, explaining that an individual $x \in X$ is of class 5 by rule r_2 (i.e. $r_2(x)$ is true) is 34.74% erroneous. As a second example, now let us have a look at rule r_8 on class 3. Its coverage is $770/798 = 96.49\%$ and the error rate is 0%. Any individual $x \in X$ such that $r_8(x)$ is of class 3, $r_8(x)$ explains the prediction of x without error. Moreover, 96.49% of the individuals of class 3 in X can be explained by r_8. All the results are presented in Table 1.

Table 1 Couples $(\tau_{max}, \varepsilon_{min})$ for every rule in the decision tree of Figure 2.

Rules	r_0	r_1	r_2	r_3	r_4	r_5	r_6
Class	5	2	5	4	2	5	1
τ_{max} (%)	100	100	98.89	96.49	100	98.89	100
ε_{min} (%)	66.94	66.44	34.74	50.31	0	0	3.41

Rules	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
Class	4	3	3	1	4	5
τ_{max} (%)	100	96.49	3.51	100	100	1.1
ε_{min} (%)	2.26	0	0	0	0	0

We propose to score the interpretability of a model M based on the following definition.

▶ **Definition 4** ((T,P,τ,ε)-interpretability). *A model M is (T,P,τ,ε)-interpretable for a class $c \in \mathcal{C}$ if there exists a set of rules $\mathcal{I} = \{r_1, \ldots, r_k\}$ on class c from M such that:*

- *Rule r_i is a (τ,ε)-rule, for any $i \in \{1, \ldots, k\}$*
- *$k \leq P$*
- *Any pair of distinct rules in $\mathcal{I}$ is disjoint*

$$T \leq \frac{|\bigcup_{i=1}^{k} ccvr(r_i)|}{|\{x \in X, \ell(x) = c\}|} \tag{6}$$

A model M is (T,P,τ,ε)-interpretable if it is (T,P,τ,ε)-interpretable for every class $c \in \mathcal{C}$.

To score the interpretability of a model M, the decision maker must set up four parameters and extracts the rules according to these parameter settings:

1. Parameter T is the global covering rate. This parameter ensures that the set of extracted rules covers a minimal amount of individuals in X labeled with the same class c. The higher T is, the more chance is that an new individual $x \in \mathcal{X}$ such that $M(x) = c$ can be explained by one of these extracted rules, hence a better interpretability.
2. Parameter P is the maximal number of rules to extract. The lower P is, the more interpretable the model is (for a given set of parameters T, τ, ε) as the set of extracted rules are then more concise.
3. Parameter τ, as explained above, is a minimal coverage rate for a rule to be part of the selection. If parameter τ is too low, the decision maker accepts to select rules that cover very few individuals with regards to the dataset X. A rule with a τ that is low may be trustful due for instance to overfitting problems.
4. Parameter ε is the maximal error rate. The higher ε is, the more error-prone, the selected rules will be.

As a first example, let us consider that $P = 1$, $T = 98\%$, $\tau = 80\%$ and $\varepsilon = 0\%$. With these settings, the decision maker is expecting that the model is highy interpretable by looking for one rule that covers most of the individuals without any error. In Figure 2, there exists such a rule, it is rule r_4 on class 2. Therefore, the DT is highly interpretable for class 2. For any individual $x \in \mathcal{X}$ such that the model predicts class 2, it can also provide the following explanation:

$$r_4(x) \text{ iff } fa(x, 3800) \leq 186.224 \wedge fa(x, 175) > 277.893 \tag{7}$$

In the context of maintenance, it simply means that the operator does not require any maintenance on the spindle as it is still in class 2 and the explanation provided by the model for this degradation class is that the current frequency amplitude at $3800Hz$ is lower than 186.224 (the spindle is not new) and the current frequency amplitude at $175Hz$ is greater than 277.893 (the spindle has not yet reached class ≥ 3). The model is as highly interpretable for class 3. The rule that covers most of the individuals of class 3 is rule r_8 but its covering rate is below $T = 98\%$, there is no rule that can be selected for class 3 based on the previous settings. The model is $(96\%, 1, 80\%, 0)$-interpretable for class 3, it is also $(100\%, 2, 3\%, 0\%)$-interpretable: by selecting rules r_8 and r_9, there is a full coverage, and it is not error-prone, however, the selection of r_9 requires a low minimal covering rate which might be considered as a suspicious explanation.

Looking now at the interpretability of the model from a global viewpoint, by setting $P = 1$, $T = 96\%$, $\tau = 80\%$ and $\varepsilon = 0\%$, it can be noticed one rule can be extracted from the tree for each class $c \in \mathcal{C}$, namely: $1 \rightarrow r_{10}, 2 \rightarrow r_4, 3 \rightarrow r_8, 4 \rightarrow r_{11}, 5 \rightarrow r_5$. Therefore, the model is $(96\%, 1, 80\%, 0)$-interpretable. However, if the decision maker looks for rules with the maximal coverage 100%, then it minimally enforces $P = 2$ and $\tau = 2\%$, the same model is therefore $(100\%, 2, 1\%, 0)$-interpretable (the extracted rules are then all the rules associated with the leaf nodes of the tree).

As this DT has been trained on a small experimental dataset, it has excellent performance due very likely to overfitting. Most of the time, rules are error-prone, hence the high interpretability of this model. Suppose now for the sake of illustration that nodes associated to rules r_7, r_8, r_{11}, r_{12} are not present in this tree. Then the model would be $(100\%, 1, 100\%, 51\%)$-interpretable on class 4. The only rule that could be used as an explanation for class 4 would likely fail to do so by providing an explanation for individuals that are actually not in class 4, hence a low interpretability of the model for class 4.

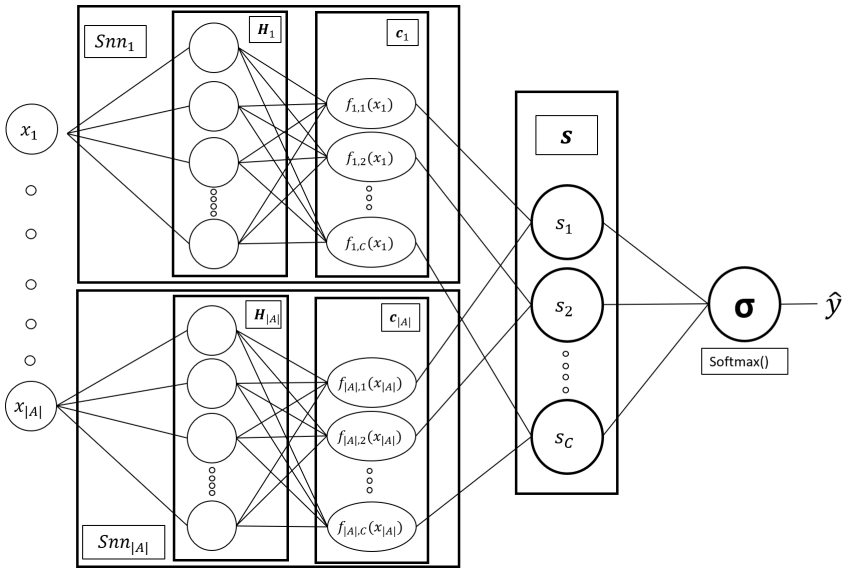


Figure 3 MNAM architecture.

5 How to score the interpretability of a MNAM?

5.1 Multi-class Neural Additive Models

There are plenty of supervised learning methods available at different scales that aim at learning interpretable models such as Linear Models, Decision Trees or Generalized Additive Models (GAMs) [16]. However, more complex and performant methods such as Multi Layer Perceptron (MLP) are usually required to learn accurate degradation models for maintenance. However, MLP are still considered as black box models so far, as the interactions between the hidden layers of the model cannot be interpreted. Recently, to overcome this issue in such models, Neural Additive Networks (NAMs) [1] have been introduced. This supervised method proposes to use the concepts of GAMs applied to neural network structures. To date, NAMs have been able to solve supervised task like regression problems or binary classification problems [21]. We have recently proposed the Multi-class Neural Additive Model (MNAM) as an extended version of the NAM algorithm for multi-class classification and applied it to solve a predictive maintenance problem [9]. The objective of this extension is to improve the interpretability of the models while keeping the benefit of a Neural Network architecture (performance, accuracy, ...). The MNAM architecture is presented in Figure 3 and briefly described here below (for more details see [9]).

A MNAM architecture is made of $|A|$ feature networks. Each feature network S_{nn_i} is composed of its input x_i, a structure H_i made up of successive hidden layers defined during the model design phase, and its output $f_i(x_i)$. The H_i structure can be composed of several hidden layers, generally made up of regular units, using a ReLU activation function. One of the problems with S_{nn} that have only one input feature is that they often struggle to approximate 1D sharp jump functions with the regular unit and a ReLU activation function on the first layer of H_i. To solve this issue, a new hidden unit, called EXp-centered-Unit (ExU), has been introduced [1] and is preferably placed in the first layer of the H_i structure. It can learn and adjust the weight parameters in logarithmic space. Each new hidden unit using an activation function σ compute $h(x)$ as follows:

$$h(x) = \sigma(e^w(x - b)). \tag{8}$$

In case that x may have negative values, these first layer units can also be replaced by ExpDive hidden units as proposed in [13]. A ExpDive hidden unit is defined by:

$$h(x) = \sigma((e^{w} - e^{-w}) \times (x - b)). \tag{9}$$

The structure of each S_{nn_i} is composed of one output $f_{i,c}(x_i)$ for each class c involved in the classification problem. The layer s of the MNAM architecture then gathers the outputs of the feature networks for each class c as a sum of these outputs:

$$s_i(x) = \sum_{i=1}^{|A|} f_{i,1}(x_i) + \beta_i \tag{10}$$

where β_i is a bias. Finally, since the network now has C outputs, a softmax function is applied to transform the layer s into a probability distribution that produces the MNAM output $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \sigma_{softmax}([s_i(x)]_{i\in\{1,\dots,C\}}) = \left[\frac{e^{s_i(x)}}{\sum_{j=1}^{C} e^{s_j(x)}}\right]_{i\in\{1,\dots,C\}} \tag{11}$$

5.2 MNAM interpretability

MNAMs are glass-box models defined in [9] which use a methodology belonging to the family of Generalized Additive Models [11] known for their ability to capture linear and non-linear relations between features and predictions while remaining globally explainable. The global explainability of MNAM mainly rely on the existence of so-called *shape functions* that can be computed once the model has been trained. They represent the exact description of the model decision process for all features [1]. The shape function of a feature a_i for the class $j \in \{1, \dots, C\}$ is given by the plot of all predictions from dataset X, that is, for any individual $x = (x_1, \dots, x_C) \in X$, the plot of $(x_i, f_{i,j}(x_i))$.

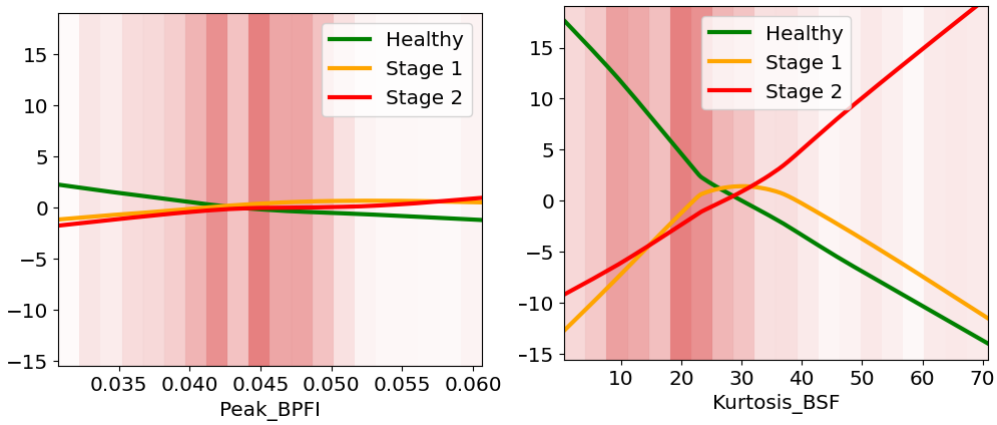


Figure 4 Example of two shape functions extracted from a Multi-class Neural Additive Model showing how the model classifies individuals with respect to the given features (namely Peak_BPFI, Kurtosis_BSF) [9].

Figure 4 illustrates a selection of two shape functions from a MNAM model that has been trained on a similar problem [9] as the one detailed in Section 3 (degradation diagnosis of bearings amongst the class *healthy*, *stage 1*, *stage 2*). The main difference here is that

instead of spectrograms of frequency amplitudes as inputs, the selected features are statistics over a set of predefined frequency values that have a physical meaning (BSF: Ball Spin Frequency, BPFI: Ball Pass Frequency Inner, ...). This figure shows how the trained model actually classifies any individual of the dataset X with respect to its Kurtosis_BSF feature ($kbsf$) and its Peak_BPFI feature ($pbpfi$). The shape functions for Kurtosis_BSF feature clearly defines a rule between the kurtosis and the degradation of the spindle bearings: for a Kurtosis_BSF within the range [0, 25], it predicts a healthy stage, for a range in [25,32] the stage 1 and for [32,72] only stage 2. In this example, this rule is simple enough for a human to understand the model's decision and is therefore interpretable. Similarly, as the shape functions associated with Peak_BPFI feature are all of them flat, it is pretty intuitive that the interpretation of the bearing's degradation does not rely on this feature.

In the aim of measuring the interpretability of such a model with the framework that is defined in Section 4, rules as defined in Definition 1 must be extracted from the shape functions. Consider for the sake of simplicity, that the diagnosis machine learning problem defined here above is only based on both features $A = \{pbpfi, kbsf\}$. For any individual $x \in X$, according to the MNAM model, a possible way to design a rule r on class $c \in \mathcal{C} = \{healthy, stage1, stage2\}$ is as follows:

$$r(x) = \{pbpfi_c(x_{pbpfi}) + kbsf_c(x_{kbsf}) = \max_{c' \in \mathcal{C}} (pbpfi_{c'}(x_{pbpfi}) + kbsf_{c'}(x_{kbsf})\}. \tag{12}$$

Intuitively speaking, such a rule r asserts that an individual x is in class c iff the sum of the shape functions for x for every feature in A is greater than any sum of the shape functions for x for another class c' (it is always the maximum by definition of the MNAM architecture, see Eq (11)).

It must be noticed that the support of such a rule is the entire set A. But the design of rules from the shape functions might be more complex to obtain finer rules with a partial feature support. For instance here, as soon as $x_{kbsf} > 3.0$, x_{pbpfi} is insignificant, so the design of a rule r only based on feature $kbsf$ is possible.

6 Conclusions and perspectives

In this paper, we have discussed the notion of interpretability of degradation models obtained by machine learning techniques. Our claim is that the level of interpretability of ML models for diagnosis should be a performance indicator as well as any other indicator like accuracy, confusion matrices.... Interpretable models, by the means of the extracted rules, provide a *causal-like* relationship between the degradation class predicted by the model and the features involved in the rules (the degradation model interprets the rule as a symptom of the degradation). The second reason why interpretability is important is obviously the confidence improvement of the human operator in charge of maintaining operating machine tools. Interpretable degradation models not only predict the current degradation class but also explains it with simple feature-based rules. While global explainability as defined by the XAI community is an absolute measure (either the model is globally explainable or not), the interpretability of each model is inherently more subjective, relying on the final decision maker, posing challenges in scoring and comparing models across different methodologies. We introduced a novel approach to quantify the level of interpretability using a framework based on four parameters defined by the decision-maker, referred to as $(T, P, \tau, \varepsilon)$-interpretability. Our results demonstrate that this rule extraction methodology can be extended to other ML techniques, such as MNAM, due to its additive structure so that the level of interpretability can be exploited to select more interpretable ML models for diagnostics.

We are currently developing degradation models using different ML techniques for a future deployment on machine-tools at Bosch-Rodez. We aim at selecting the best tradeoff between accuracy performance and level of interpretability for the deployed model. Our immediate focus will be on enhancing the algorithm to incorporate the entire rule set from a model to accurately determine its $(T, P, \tau, \varepsilon)$-interpretability score. Subsequently, we aim to generalize rule extraction across the diverse machine learning methods we use to enable a universally applicable interpretability scoring system, independent of the underlying method.

References

1 Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4699–4711. Curran Associates, Inc., 2021.

2 Maryam Ahang, Mostafa Abbasi, Todd Charter, and Homayoun Najjaran. Condition monitoring with incomplete data: An integrated variational autoencoder and distance metric framework. In *IEEE 20th International Conference on Automation Science and Engineering*, Bari, Italy, 2024.

3 Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995. Knowledge-based neural networks. doi:10.1016/0950-7051(96)81920-4.

4 Mohammad Azad, Igor Chikalov, Shahid Hussain, Mikhail Moshkov, and Beata Zielosko. Decision rules derived from optimal decision trees with hypotheses. *Entropy*, 23(12):1641, 2021. doi:10.3390/E23121641.

5 Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. Interpretable random forests via rule extraction. In *International Conference on Artificial Intelligence and Statistics*, pages 937–945. PMLR, 2021. URL: http://proceedings.mlr.press/v130/benard21a.html.

6 Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

7 Xiaohan Chen, Beike Zhang, and Dong Gao. Bearing fault diagnosis base on multi-scale cnn and lstm model. *Journal of Intelligent Manufacturing*, 32:971–987, 2021. doi:10.1007/S10845-020-01600-2.

8 Victor Contreras, Niccolo Marini, Lora Fanda, Gaetano Manzo, Yazan Mualla, Jean-Paul Calbimonte, Michael Schumacher, and Davide Calvaresi. A dexire for extracting propositional rules from neural networks via binarization. *Electronics*, 11(24), 2022.

9 Charles-Maxime Gauriat, Yannick Pencolé, Pauline Ribot, and Gregory Brouillet. Multi-class Neural Additive Models : An Interpretable Supervised Learning Method for Gearbox Degradation Detection. In *2024 IEEE International Conference on Prognostics and Health Management*, Spokane , WA, United States, June 2024. URL: https://hal.science/hal-04562531.

10 Charles-Maxime Gauriat, Yannick Pencolé, Pauline Ribot, and Gregory Brouillet. An experimental setup for learning models for the rul estimation of bearings in rotary machine tools: towards the learning of more interpretable models. Technical report, LAAS-CNRS, 2023. URL: https://hal.science/hal-04715967.

11 Trevor Hastie and Robert Tibshirani. Generalized Additive Models. *Statistical Science*, 1(3):297–310, 1986. doi:10.1214/ss/1177013604.

12 Carlos Hernández-Espinosa, Mercedes Fernández-Redondo, and Mamen Ortiz-Gómez. Rule extraction from a multilayer feedforward trained network via interval arithmetic inversion. In José Mira and José R. Álvarez, editors, *Computational Methods in Neural Modeling*, pages 622–629, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-44868-3_79.

13 Minkyu Kim, Hyun-Soo Choi, and Jinho Kim. Higher-order neural additive models: An interpretable machine learning model with feature interactions. *Arxiv*, September 2022. doi:10.48550/arXiv.2209.15409.

14 Zachary C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, June 2018.

15 Brian Liu and Rahul Mazumder. Fire: An optimization approach for fast interpretable rule extraction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1396–1405, 2023. doi:10.1145/3580305.3599353.

16 Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Knowledge Discovery and Data Mining*, 2012.

17 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017. arXiv:1705.07874.

18 Ricards Marcinkevics and Julia Vogt. Interpretability and explainability: A machine learning zoo mini-tour, December 2020.

19 Vincent Margot and George Luta. A new method to compare the interpretability of rule-based algorithms. *AI*, 2(4):621–635, 2021.

20 Joao Marques-Silva and Alexey Ignatiev. No silver bullet: interpretable ml models must be explained. *Frontiers in artificial intelligence*, 6:1128212, April 2023.

21 Samad Moslehi, Hossein Mahjub, Maryam Farhadian, Ali Soltanian, and Mojgan Mamani. Interpretable generalized neural additive models for mortality prediction of covid-19 hospitalized patients in hamadan, iran. *BMC Medical Research Methodology*, 22, December 2022.

22 Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego,CA,USA, June 2016.

23 Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019. doi:10.1038/S42256-019-0048-X.

24 Victor Feitosa Souza, Ferdinando Cicalese, Eduardo Laber, and Marco Molinaro. Decision trees with short explainable rules. *Advances in neural information processing systems*, 35:12365–12379, 2022.

25 Sergey Yarushev, Alexey Averkin, and Vladimir Kosterev. Rule extraction methods from neural networks. In *Recent Trends in Intelligence Enabled Research*, pages 1–9. Springer Nature Singapore, 2023.

26 Qing Zhou, Fenglu Liao, Chao Mou, and Ping Wang. Measuring interpretability for different types of machine learning models. In Mohadeseh Ganji, Lida Rashidi, Benjamin C. M. Fung, and Can Wang, editors, *Trends and Applications in Knowledge Discovery and Data Mining*, pages 295–308, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-030-04503-6_29.

27 Zhi-Hua Zhou. Rule extraction: Using neural networks or for neural networks? *Journal of Computer Science and Technology*, 19(2):249–253, March 2004. doi:10.1007/BF02944803.

28 Jan Zilke, Eneldo Mencía, and Frederik Janssen. Deepred – rule extraction from deep neural networks. In *Discovery Science*, pages 457–473, October 2016.

Test Selection for Diagnosing Multimode Systems

Mattias Krysander
Department of Electrical Enginering, Linköping University, Sweden

Fatemeh Hashemniya
Department of Electrical Enginering, Linköping University, Sweden

Abstract

This work considers the problem of selecting residuals for consistency-based diagnosis of multimode systems. The system operation mode is assumed to be given by a set of known discrete variables. The number of operation modes grows exponentially with the number of binary variables, thus methods enumerating the modes are not feasible. Here a method is proposed to select a small subset of residuals for diagnosing multimode systems. The selection is based on the fault signature of the residuals for the different modes of operation. To avoid the exponential growth of the number of modes, the multimode fault signature matrix is used to compute the diagnosability of the residuals. The approach is inspired and exemplified by a dynamically configurable battery pack. The result is a small set of residuals with the maximum diagnosability in all operation modes.

2012 ACM Subject Classification Hardware → Online test and diagnostics; Hardware → Batteries

Keywords and phrases Consistency-based Diagnosis, Residual Selection, Multimode Systems, Battery Application

Digital Object Identifier 10.4230/OASIcs.DX.2024.28

Category Short Paper

Supplementary Material
Software: `https://github.com/MattiasKrysander/Multimode-Test-Selection`

Funding *Mattias Krysander*: Swedish Electromobility Center, Energimyndigheten.
Fatemeh Hashemniya: Swedish Electromobility Center, Energimyndigheten.

1 Introduction

Residual selection is crucial in diagnosis system design using pre-compiled tests for detecting and isolating faults. The selection of residuals is a combinatorial problem, and the number of possible combinations grows exponentially with the number of residuals. Furthermore, the number of residuals grows exponentially in the redundancy of the model. The problem is even more challenging in multimode systems where the system can operate in different operational modes. Thus, there is a need to develop computationally efficient algorithms to select a small set of residuals that can detect and isolate faults in all operation modes. Set minimal and minimum cardinality solutions could be desirable but due to the highly combinatorial nature of the problem, it is not always possible to find optimal solutions. Here we propose a greedy algorithm to select a small set of residuals for diagnosing multimode systems with maximum diagnosability in all operation modes. The selection is based on a fault signature matrix that describes the fault influence on the residuals for the different operation modes.

Structured residuals and fault signature matrices are discussed in [8], where two isolation patterns are introduced: weakly and strongly isolating structures. Weakly isolating structures can isolate faults when the column-matching approach is applicable, whereas strongly isolating structures can isolate faults using the consistency-based approach. In the presence of non-ideal residuals, such as those caused by model errors and uncertainties, only strongly isolating


© Mattias Krysander and Fatemeh Hashemniya;
licensed under Creative Commons License CC-BY 4.0



structures are applicable, which is the approach adopted in this work. Furthermore,[17] establishes a connection between the fault signature matrix used in the Fault Detection and Isolation (FDI) community and consistency-based diagnosis performed in the DX community. The foundation of the present work aligns with these principles. A recent comprehensive review of control system structural analysis diagnosability is provided in [9].

The problem of test selection for diagnosing single-mode systems has been studied in the literature. For instance, [3] formulates the test selection problem as an integer programming problem. Additionally, research has explored the combination of sensor placement and residual selection, as discussed in [16]. This problem has been formulated and solved as a mixed-integer optimization problem in [14]. Residual selection has also been studied in the context of distributed systems. For example, [13] addresses this problem using binary integer linear programming. Additionally, multimode systems have been considered in the literature [1, 12]. In [1], a solution to the double challenge of system decomposition and diagnostic test selection, aiming to minimize subsystem interconnections while maximizing diagnosability is presented. In [12], an online method is proposed to detect mode changes and select residuals dynamically based on the current mode hypothesis. In contrast to the latter work, the approach adopted in this work is offline, where residuals are selected before system operation. In [15], a reconfigurable battery system similar to the one employed in this paper is presented, but it utilizes a different heuristic approach to manage the complexity of mode switches. Test selection methods for single-mode systems have been developed to evaluate residual fault sensitivity using training data, enabling robust residual selection. Examples of such approaches can be found in [5] and [11]. However, these works do not consider systems with multimode operation.

The contributions of this work is an algorithm to compute the diagnosability of residuals in multimode systems and for selecting a small set of residuals for diagnosing multimode systems with maximum diagnosability in all operation modes. A Python package is developed and is available at `https://github.com/MattiasKrysander/Multimode-Test-Selection`. It is based on `dd` library [4] for efficiently handling Boolean functions and logic inference. The methods are based on the fault signatures of the residuals for the various modes of operation, without accounting for the quantitative impact of faults. The approach is not limited to any specific residual generation method, both model-based and data-driven residuals can be used if the fault signature of the residual for the different operation modes is known. The fault influence and decoupling are important since the diagnosability is derived from a consistency-based diagnosis framework. Since the selection method relies only on the qualitative fault signatures of the residuals, structural methods can also be applied to compute potential residuals along with their expected fault influence. This enables residual selection based on the potential residuals, followed by the development of only the necessary residual generators as determined by the selection process. The approach is inspired by and demonstrated through a modular, dynamically configurable battery pack. This kind of battery pack has all the challenges that make the problem interesting, such as multiple operation modes, many sensors making the model redundancy large, and many components that need supervision. The selection of residuals is non-trivial because the number of residuals to choose from is large and the number of operation modes is exponential in the number of binary operation mode variables. The result is a small set of residuals with the maximum diagnosability in all operation modes.

The residual selection problem is formally defined in Section 2. A motivating case study of a modular dynamically reconfigurable battery pack is presented in Section 3 where multimode residuals are exemplified. Section 4 introduces the concept of fault signature of residuals in

multimode systems. Then the diagnosability of residuals in multimode systems is discussed in Section 5. The residual selection algorithm is presented in Section 6. The algorithm's results applied to the battery pack case study are presented in Section 7 and the algorithm's computational complexity is discussed in Section 8. Finally, the paper is concluded in Section 9.

2 Problem Formulation

The work considers a system with a set of faults F. The system can operate in different modes and the set of all valid operation modes is $\mathbb{M}$, not to be confused with fault modes. This work assumes that the operation mode is measured or controlled. The system is monitored by sensors that produce residuals by applying them to a set of residual generators, R. The residuals are used to detect and isolate faults. The applicability of the residuals and their fault influence depends on the system's operation mode. The fault influence of the residuals is represented by the fault signature matrix, S_R, where each entry (r_j, f_i) is a Boolean function over the set of valid operation modes $\mathbb{M}$. This function evaluates to $\mathbf{T}$ (true) if residual r_j is sensitive to the fault f_i in mode m, and to $\mathbf{F}$ (false) otherwise.

Maximum fault diagnosability is achieved by utilizing all residuals R. However, all residuals are typically not needed for maximum diagnosability, here detectability and single-fault isolability will be considered. The challenge lies in selecting an appropriate subset, given the many possibilities, which grow exponentially with the number of residuals. This challenge becomes even more difficult in multimode systems, where the complexity of the problem increases significantly. The main problem addressed in this paper is how to identify a small subset of residuals, $R_s \subseteq R$, with the same diagnosability as all residuals across all operation modes.

3 Case Study: A Multimode Modular Battery Pack

A motivating case study that will be used to illustrate concepts and results of the paper is a multimode modular dynamically reconfigurable battery pack. It includes n modules in series, numbered from 1 to n, as illustrated in Fig. 1. Module k in Fig. 1(a) includes a battery cell, modeled as an equivalent circuit model, along with a full-bridge converter consisting of four MOSFET switches, i.e., S_1-S_4, and two sensors, i.e., a voltage and a current sensor, measuring $v_{\text{cell},k}$ and $i_{\text{cell},k}$, respectively. Two pack sensors measure the output voltage v_{pack} and the output current i_{pack}, respectively, as shown in Fig. 1(b). Faults for each battery cell and each sensor will be considered.

The term multimode describes the system's ability to operate in different modes, determined by different switch positions. Each module has three valid modes depending on how its cell is connected to the battery pack. In forward mode, the cell adds to the pack voltage, in backward mode it subtracts, and in bypass mode, the cell is disconnected from the circuit. The module is said to be turned on if it is in forward or backward mode.

A model of the battery system has been implemented in the Fault Diagnosis Toolbox (FDT) [6, 7]. The model of module j is

```
v_p_der_j == i_cell_j / Cp_j - v_p_j / (Rp_j * Cp_j),...
v_ocv_j == OCV_fun( SOC_j ),...
SOC_der_j == -1/(3600*Q_j)*i_cell_j,...
DiffConstraint("SOC_der_j","SOC_j"),...
v_cell_j == v_p_j + R0_j * i_cell_j + v_ocv_j + f_cellj,...
DiffConstraint("v_p_der_j","v_p_j"),...
```

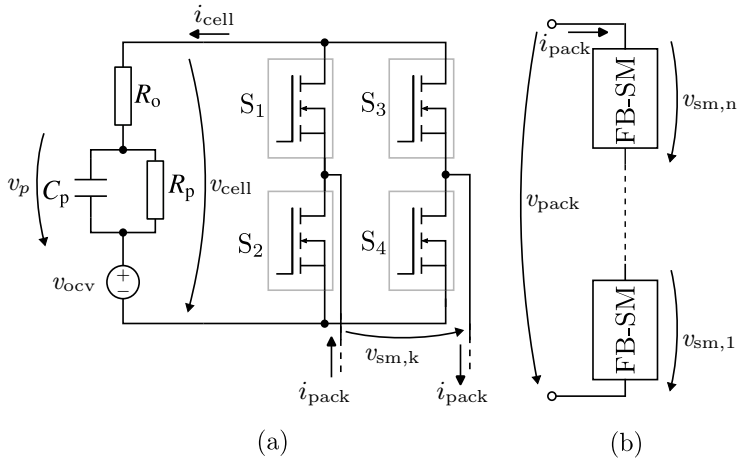


Figure 1 A modular dynamically reconfigurable battery system. (a) A battery pack with n modules. (b) A battery module.

```
v_sm_j == v_sm_fun( forward_j, backward_j, v_cell_j),...
i_cell_j == i_cell_fun( forward_j, backward_j, i_pack),...
y_i_j == i_cell_j + f_ij,...
y_v_j == v_cell_j + f_vj
```

The battery is modeled in rows 1-6 where `i_cell_j` is the cell current, `v_cell_j` the cell voltage, and `SOC_j` its state-of-charge. The open circuit voltage of the cell is interpolated from its state-of-charge in function `OCV_fun`. The full-bridge converter is modeled in rows 7-8 where `v_sm_j` is the output voltage of the module and `i_pack` is the battery pack current. The Boolean variables `forward_j` and `backward_j` determine the operation mode of the module. For example, the cell current and pack current are related by the function `i_cell_fun` given by

```
function i_cell = i_cell_fun( forward, backward, i_pack)
if forward
    i_cell = i_pack;
elseif backward
    i_cell = -i_pack;
else
    i_cell = 0;
end
```

The voltage of the module `v_sm_j` is governed by a similar function `v_sm_fun`. In addition to the modules, the battery pack has sensors measuring the total output voltage and current of the pack

```
v_pack == v_sm_1 + v_sm_2 + ... + v_sm_n
y_v_pack == v_pack + f_vpack
y_i_pack == i_pack + f_ipack
```

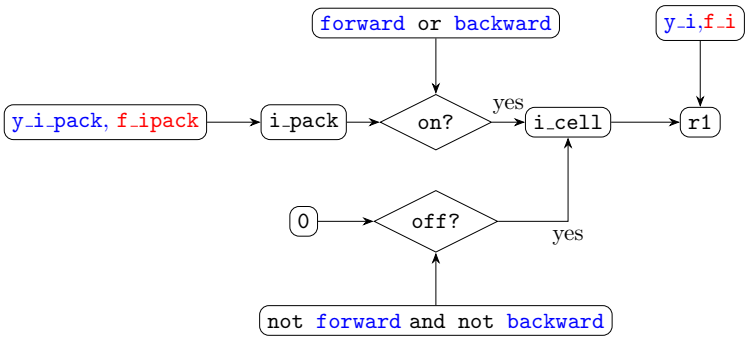


Figure 2 Graph representation of the residual generation function `ResGen_1`. In particular it shows the influence of the faults `f_i` and `f_ipack` on residual `r1`.

For each module j, faults for the cell `f_cellj`, current sensor `f_ij`, and voltage sensor `f_vj` are considered as well as for the pack voltage and current sensors `f_vpack` and `f_ipack` respectively. In the FDT, faults are represented as general fault signals, although they can equivalently be interpreted as assumptions on the components. For instance, `y_i_pack == i_pack + f_ipack` could also be interpreted as

$$\text{OK(Pack current sensor)} \rightarrow \texttt{y_i_pack == i_pack}. \tag{1}$$

4 Fault Signature of Residuals

The input to the residual selection algorithm proposed in this work is a multimode extension of the fault signature matrix. The underlying concept will be introduced and examined in detail in this section.

Consider a one-module battery pack with a pack current sensor to illustrate the concept of residuals and fault signature. In this example, no sensor for measuring pack voltage is included. Using the FDT, three multimode residuals are generated for the battery pack. One residual `r1` is comparing the current measurements as follows

```
function [r1] = ResGen_1(y_i,y_i_pack,forward,backward)
i_pack = y_i_pack;
i_cell = y_i;
r = i_cell - i_cell_fun(forward, backward, i_pack);
```

A computational graph is shown in Figure 2, where known inputs are in blue and faults are in red. The fault `f_i` will influence the residual independent of the mode. The fault `f_ipack` will only influence the residual if the module is turned on.

Next, the fault signature of a residual in a multimode system is formally defined.

▶ **Definition 1** (Multimode fault signature of a residual). *In a multimode system, the signatures of fault f of residual r is a Boolean function $S_r^f : \mathbb{M} \rightarrow \mathbb{B}$ of operation modes such that it is* **T** *if fault f influences residual r in mode $m \in \mathbb{M}$ and* **F**, *otherwise. The* multimode fault signature of residual r *is the collection of the fault signatures of all faults F in the system, i.e., $S_r = \{S_r^f | f \in F\}$.*

For residual `r1` in Figure 2, the multimode fault signature is given by $S_{\texttt{r1}}^{\texttt{f_ipack}} = \texttt{forward} \vee \texttt{backward}$ and $S_{\texttt{r1}}^{\texttt{f_i}} = \mathbf{T}$. The fault signature of a residual does not depend on whether a module is connected in the forward or backward mode. It is only dependent on if the module

Table 1 The multimode fault signature matrix of the residuals for the dynamically configurable battery pack with one module and a pack current sensor.

residual	f_ipack	f_i	f_cell	f_v
1	on	**T**	**F**	**F**
2	**F**	**T**	**T**	**T**
3	on	**F**	**T**	**T**

is turned on. Since the battery example will be used throughout the paper, we will simplify the notation by introducing a Boolean variable $\texttt{on} = \texttt{forward} \vee \texttt{backward}$ such that the fault signature can be written as $S_r^{\texttt{f_ipack}} = \texttt{on}$.

▶ Remark. Note that the fault signature of a residual is not dependent on the gain from the fault to the residual, it is only a Boolean function indicating if the fault influences the residual. For example, the sign of the fault sensitivity of `f_ipack` is different in forward and backward directions but the fault signature is the same.

As mentioned before, in a single mode case the fault signature of a set of residuals R is often collected in a so-called fault signature matrix with rows and columns corresponding to residuals and faults respectively. An entry (r_i, f_j) is either **T** if fault f_j influences residual r_i or **F** otherwise. The signature can depend on the system operation mode in the multimode case and is extended to multimode systems in the following definition.

▶ **Definition 2** (Multimode fault signature matrix). *Given a system of valid operation modes $\mathbb{M}$, a set of faults F, and a set of residuals R; the* multimode fault signature matrix *is defined as a matrix S_R where each row corresponds to a residual $r \in R$ and each column corresponds to a fault $f \in F$ and the element in position (r, f) is S_r^f.*

The multimode fault signature will also be evaluated for subsets of residuals $R_s \subseteq R$ and then the notation S_{R_s} will be used.

The fault signature matrix of the three residuals derived for the one-module battery-pack is given in Table 1. The first residual is the one illustrated in Figure 2.

Residual selection will be based on fault signature matrices similar to the one in Table 1. The fault signature matrix will be used to evaluate the diagnosability of different subsets of residuals which will be the topic of the next section.

5 Fault Diagnosability of Residuals

This section explains how to compute the multimode diagnosability of a residual set given their fault signature matrix. The diagnosability will be defined as the ability to detect and isolate faults using the residuals which depends on how the residuals are integrated into diagnosis computations. This work is based on a consistency-based framework, thus adopting the following definition of diagnosis from [2].

▶ **Definition 3** (Consistency-based diagnosis). *Given a diagnostic model $\mathcal{M}$ and observations $\mathcal{O}$, a diagnosis is an assignment $\mathcal{D}$ of a behavioral mode to each considered fault such that $\mathcal{M} \cup \mathcal{O} \cup \mathcal{D}$ is consistent. A diagnosis is minimal if it is minimal considering the set of faults.*

Here the diagnostic model is the set of residuals R with the corresponding fault signature matrix S_R. Observations are the residuals that trigger alarms $R_a \subseteq R$ and the present mode of operation $m \in \mathbb{M}$. A behavioral mode is represented as subsets of present faults $F_b \subseteq F$. For this particular case, the diagnosis is given by the following proposition.

▶ **Proposition 4** (Diagnosis of multimode residuals). *Given a fault signature matrix S_R and the observations (R_a, m) a behavioral mode $F_b \subseteq F$ is a diagnosis if it is a minimal hitting set of the sets $\{f|S_r^f(m) = \mathbf{T}\}$ for all $r \in R_a$. A diagnosis is minimal if no proper subset is a diagnosis.*

In the diagnosability analysis, the best possible residual response is considered. Then the diagnosability result will be an overestimate of the true diagnosability.

▶ **Definition 5** (Ideal fault response). *A residual r has an ideal fault response for all modes $m \in \mathbb{M}$ if it triggers an alarm for the set of faults $\{f|S_r^f(m) = \mathbf{T}\}$ it is influenced by according to the fault signature matrix.*

A consequence of ideal fault response and the definition of diagnosis is that F_b is a diagnosis if F_b is the present fault mode. Even if it is possible to take multiple faults into account, we will in the continuation consider the single fault case for ease of notation. This is a common assumption using pre-compiled tests. The no-fault mode will be denoted by `NF` and a single fault f_i with a slight abuse of notation.

▶ **Definition 6** (Structural detectability and isolability of multimode residuals). *A single fault f is structurally detectable in operation mode m with a set of residuals R if f is a minimal diagnosis given that all residuals have an ideal fault response. A fault f_i is structurally isolable from a fault $f_j (i \neq j)$, in mode m with a set of residuals R if f_i is a minimal diagnosis but not f_j given that all residuals have ideal fault response.*

To illustrate the implication of this definition on residual selection, consider a single-mode system with fault signature matrix

	f_1	f_2	f_3
r_1	**T**	**F**	**T**
r_2	**F**	**T**	**T**
r_3	**T**	**T**	**F**

(2)

The column-matching approach matches the residual response with the columns in the signature matrix to find the diagnoses. Then it would be sufficient to use the first two residuals to isolate all faults since all faults will have a unique signature. This isn't enough in the consistency-based approach as will be shown next. Assume that fault f_1 is present. Then residual r_1 will be triggered and the single faults f_1 and f_3 will be diagnosed. Hence, f_1 is not structurally isolable from f_3 with the first two residuals. Full single-fault isolability is obtained using all residuals.

Detectability and isolability will be unified in the following definition of diagnosability which is the main concept of this work.

▶ **Definition 7** (Structural diagnosability of multimode residuals). *Let the structural diagnosability be a Boolean function $D_R(f_i, f_j) : \mathbb{M} \to \mathbb{B}$, such that*

$$D_R(f, \texttt{NF})(m) = \begin{cases} \mathbf{T} & \text{if } f_i \text{ is structurally detectable in mode } m \\ & \quad \text{with ideal fault respons of } R \\ \mathbf{F} & \text{otherwise} \end{cases} \tag{3}$$

$$D_R(f_i, f_j)(m) = \begin{cases} \mathbf{T} & \text{if } f_i \text{ is structurally isolable from } f_j (i \neq j) \text{ in mode } m \\ & \quad \text{with ideal fault response of } R \\ \mathbf{F} & \text{otherwise} \end{cases} \tag{4}$$

The following theorem shows how the fault signature matrix can be used to compute the diagnosability of a set of residuals.

▶ **Theorem 8.** *Given a fault signature S_R for a set of multimode residuals R, the structural detectability of the residuals in R is given by*

$$D_R(f_i, \texttt{NF}) = \bigvee_{r \in R} S_r^{f_i} \tag{5}$$

and the structural isolability by

$$D_R(f_i, f_j) = \bigvee_{r \in R} \left(S_r^{f_i} \wedge \neg S_r^{f_j} \right). \tag{6}$$

Proof. It is sufficient to consider an arbitrary fault pair (f_i, f_j). We will start to prove (6).

Select an arbitrary $m \in \mathbb{M}$ such that $D_R(f_i, f_j)(m) = \mathbf{T}$. Then there exists a residual $r \in R$ such that $S_r^{f_i}(m) = \mathbf{T}$ and $S_r^{f_j}(m) = \mathbf{F}$. This implies that $\bigvee_{r \in R} \left(S_r^{f_i} \wedge \neg S_r^{f_j} \right)(m) = \mathbf{T}$, thus $D_R(f_i, f_j) \models \bigvee_{r \in R} \left(S_r^{f_i} \wedge \neg S_r^{f_j} \right)$.

Now, select an arbitrary $m \in \mathbb{M}$ such that $\bigvee_{r \in R} \left(S_r^{f_i} \wedge \neg S_r^{f_j} \right)(m) = \mathbf{T}$. Then there exists a residual $r \in R$ such that $S_r^{f_i}(m) = \mathbf{T}$ and $S_r^{f_j}(m) = \mathbf{F}$. This implies that $D_R(f_i, f_j)(m) = \mathbf{T}$, thus $\bigvee_{r \in R} \left(S_r^{f_i} \wedge \neg S_r^{f_j} \right) \models D_R(f_i, f_j)$ and (6) is proved.

The proof of (5) follows from (6) by noting that no residual will trigger an alarm in NF, i.e., $S_r^{\emptyset} \equiv \mathbf{F}$ for all $r \in R$. Formula (5) follows by letting $f_j = \emptyset$ in (6) and the proof is complete. ◀

To exemplify diagnosibility matrices, consider the fault signature matrix in Table 1. The diagnosability of the residual 1 is according to Theorem 8 given by

	NF	f_ipack	f_i	f_cell	f_v
f_ipack	on	**F**	**F**	on	on
f_i	**T**	¬on	**F**	**T**	**T**
f_cell	**F**	**F**	**F**	**F**	**F**
f_v	**F**	**F**	**F**	**F**	**F**

(7)

and the diagnosability of all 3 residuals is given by

	NF	f_ipack	f_i	f_cell	f_v
f_ipack	**on**	**F**	**on**	**on**	**on**
f_i	**T**	**T**	**F**	**T**	**T**
f_cell	**T**	**T**	**T**	**F**	**F**
f_v	**T**	**T**	**T**	**F**	**F**

(8)

The detectability is given by the column NF in blue and the remaining part specifies the isolability. Full single fault diagnosability is given by a matrix with **T** in all entries except for the diagonal entries (f_i, f_i) which are **F**.

The diagnosability matrix (8) shows that the 3 residuals have full single fault diagnosability except for the following two exceptions. The entries on in the first row show that fault f_ipack is not detectable nor isolable from any other fault in mode on $= \mathbf{F}$. If on $= \mathbf{T}$ the fault is detectable and uniquely isolable. The two by two block in pink with entries **F** shows that f_cell and f_v are not isolable from each other in any mode.

6 Test Selection for Multimode Systems

This section describes the propsed algorithm to select a small subset of residuals with the same diagnosability as all residuals. The algorithm is based on the diagnosability of the set of residuals. The basic idea is to start with the empty set of residuals and then iteratively add the residual that improves the diagnosability the most.

Prior to presenting the algorithm, some convenient notation is introduced. Let the diagnosability matrix for a residual set R be denoted D_R. Furthermore, let $\neg D_R$ denote the elementwise negation of D_R. This means that $\neg D_R$ represents the diagnosability properties not provided by the residuals in R. Let g be a function from the set of diagnosability matrices to a scalar-valued goodness measure. A residual with the highest goodness measure will be selected. Different choices of improvement functions g will be discussed after the algorithm. The algorithm is described in Algorithm 1. The `dd` library [4] is used for efficient implementation of the fault signature and diagnosability matrices. Enumeration of all possible modes is avoided by encoding operation modes by Boolean functions.

Algorithm 1 Test Selection Algorithm.

Input: Fault signature matrix S_R for residuals R.
Output: Selected residuals $R_s \subseteq R$.
Initialize: Let the selected residuals be $R_s = \emptyset$ and the remaining residuals $R_r = R$.
while $R_r \neq \emptyset$ **do**
  for each remaining residual $r \in R_r$ **do**
    Compute diagnosability improvement of adding r to R_s: $\Delta D^r_{R_s} = D_{\{r\}} \wedge \neg D_{R_s}$.
  end for
  if no improvement is possible, i.e., $\max_{r \in R_r} g(\Delta D^r_{R_s}) = 0$ **then**
    break
  else
    Select a residual r^* with the largest improvement, i.e., $r^* = \arg\max_{r \in R_r} g(\Delta D^r_{R_s})$.
    Update the residual sets: $R_s = R_s \cup \{r^*\}$, $R_r = R_r \setminus \{r^*\}$.
  end if
end while

6.1 Improvement Functions

Two different improvement functions are proposed. The first counts the number of improved entries in the diagnosability matrix achieved by adding a residual. The diagnosability improvment of adding residual r to the set R_s is $\Delta D^r_{R_s}$. The improvement function g is in this case defined as

$$g(\Delta D^r_{R_s}) = |\{(f_i, f_j) \in (F \cup \texttt{NF}) \times F | \Delta D^r_{R_s}(f_i, f_j) \not\equiv \mathbf{F}\}|. \tag{9}$$

Algorithm 1 with improvement function (9) will be called `TestSelection`.

To give an example of how the improvement function is used in the residual selection consider the one-module battery pack and assume that no residuals have been selected, i.e., $R_s = \emptyset$. Consider the improvement function g for the first residual in Table 1. All entries of the diagnosability matrix of the empty set of residuals are $D_\emptyset(f_i, f_j) \equiv \mathbf{F}$. The diagnosability improvemnet $\Delta D^{\texttt{r1}}_\emptyset = D_{\{\texttt{r1}\}}$ which is given in (7). The improvement of adding residual 1 is given by the number of entries in the diagnosability matrix different from $\mathbf{F}$ which is 7. The improvement of adding residual 2 is 6 and adding residual 3 is 8. Hence the third residual is selected first.

The second improvement function will contrary to the function defined in (9) take into account the number of operation modes for which diagnosability is improved. Then the improvement function is defined as the number of assignments as

$$g(\Delta D^r_{R_s}) = \sum_{(f_i,f_j)\in(F\cup\mathtt{NF})\times F} |\{m|\Delta D^r_{R_s}(f_i,f_j)(m) = \mathbf{T}\}|. \tag{10}$$

Algorithm 1 with improvement function (10) will be called `TestSelectionAssignment`.

Consider again the improvement function for adding residual 1 to the empty set of residuals. The diagnosability improvement is given in (7). With the introduced notation, there are 2 operation modes of the system, it is either turned on, i.e., $\mathtt{on} = \mathbf{T}$ or turned off, i.e., $\mathtt{on} = \mathbf{F}$. The evaluation of (10) is, to sum up the contributions from each entry. Entries that are $\mathbf{T}$ count as 2, entries that are $\mathbf{F}$ count as 0, and the rest count as 1. Thus, the improvement adding residual 1 is 10. The improvement adding residual 2 is 12 and residual 3 is 8, thus the second residual is selected. The computational complexity will be discussed in Section 8.

6.2 Diagnosability in any Operation Mode

For systems that frequently cycle through all operation modes or systems where the operation mode can be selected partially based on diagnosis requirements, it is possible to relax the diagnosability goal to possibly reduce the number of residuals. It could be sufficient to aim for maximum diagnosability first after cycling through all operation mode. The problem is formally stated as follows. The goal is to select a small set of residuals $R_s \subseteq R$ such that for each diagnosability property (f_i, f_j) satisfying $D_R(f_i, f_j) \not\equiv \mathbf{F}$ there exists a mode $m \in \mathbb{M}$ such that

$$D_{R_s}(f_i, f_j)(m) = \mathbf{T} \tag{11}$$

This problem is solved by Algorithm 1 with the improvement function `TestSelection` defined in (9) by using a modified fault influnce matrix $\tilde{S}_R$ defined as

$$\tilde{S}_R(f_i, f_j) \equiv \begin{cases} \mathbf{F} & \text{if } S_R(f_i, f_j) \equiv \mathbf{F} \\ \mathbf{T} & \text{otherwise.} \end{cases} \tag{12}$$

Algorithm 1 with improvement function (9) where the modified fault influnce matrix $\tilde{S}_R$ is used will be called `TestSelectionAnyMode`.

7 Method Demonstration

The proposed method is demonstrated on a modular battery pack. Consider the modular battery pack described in Section 3 with 2 modules and sensors measuring pack current and voltage. Multimode residuals have been generated and the fault signature matrix of those are given in Table 2. Two operation mode variables `on1` and `on2` indicate if the corresponding module is on. The residuals are generated using the Fault Diagnosis Toolbox [6, 7] and the enumeration of residuals corresponds to the underlying minimal structurally overdetermined sets. The fault signature matrix is generated using the method described in Section 5. The residuals are partitioned into sets relating to the same set of modules. The first three residuals, residual 1-3, only involve module 2, residuals 4, 11, and 12 only module 1, and the

Table 2 The multimode fault signature matrix for the residuals is derived from the dynamically configurable battery pack with 2 modules and sensors measuring pack current and voltage.

Residual	f_cell1	f_i1	f_v1	f_cell2	f_i2	f_v2	f_vpack	f_ipack
1	**F**	**F**	**F**	**F**	**T**	**F**	**F**	on2
2	**F**	**F**	**F**	**T**	**T**	**T**	**F**	**F**
3	**F**	**F**	**F**	**T**	**F**	**T**	**F**	on2
4	**F**	**T**	**F**	**F**	**F**	**F**	**F**	on1
11	**T**	**T**	**T**	**F**	**F**	**F**	**F**	**F**
12	**T**	**F**	**T**	**F**	**F**	**F**	**F**	on1
7	**F**	**F**	on1	**F**	**F**	on2	**T**	**F**
8	**F**	**F**	on1	on2	on2	**F**	**T**	**F**
9	**F**	**F**	on1	on2	**F**	**F**	**T**	on2
15	on1	on1	**F**	**F**	**F**	on2	**T**	**F**
18	on1	**F**	**F**	**F**	**F**	on2	**T**	on1
16	on1	on1	**F**	on2	on2	**F**	**T**	**F**
17	on1	on1	**F**	on2	**F**	**F**	**T**	on2
20	on1	**F**	**F**	on2	on2	**F**	**T**	on1
21	on1	**F**	**F**	on2	**F**	**F**	**T**	on1∨on2

Table 3 Test selection applied to the two-module battery pack.

Algorithm	**Selected residuals**	**Improvement**
`TestSelection`	16, 3, 12, 1, 4, 7	32, 20, 20, 10, 10, 8
`TestSelectionAnyMode`	16, 3, 12, 1, 4, 7	32, 15, 11, 2, 2, 2
`TestSelectionAssignment`	2, 11, 21, 7, 1, 4, 3, 12	72, 72, 50, 15, 13, 10, 4, 4

rest include both modules. The order of the rows in the fault signature matrix can change the result of the residual selection because if there are residuals with equal improvement, the first one will be selected.

Table 3 shows the result of applying the different versions of test selection. The residuals are listed in the selection order together with the value of the improvement function for each selection. For this example the smallest solution found contains 6 residuals and the same solution is found both for the `TestSelection` and `TestSelectionAnyMode` algorithms. The `TestSelectionAssignment` algorithm selects 8 residuals, where residuals 2 and 11 are chosen first prioritizing residuals that improve the diagnosability in the most operation modes.

The diagnosability matrix for both the set of all residuals and the selected ones in Table 3 is given in Table 4. The diagnosability matrix shows that full single fault diagnosability with the exceptions that `f_ipack` is detectable and uniquely isolable if any module is turned on, and the cell fault `f_celli` and corresponding voltage sensor fault `f_vi` is isolable from each other if the corresponding module is turned on.

Diagnosability analysis can be applied to the model defined in Section 3 with the method proposed in [10]. The diagnosability of the model and the selected residuals are equal. This means that maximum diagnosability for all modes is achieved with the 6 residuals. The fault signature matrix of these residuals is given in Table 5. Two local residuals for each module and two residuals including both modules are selected to achieve maximum diagnosability. All the residuals are defined in all operation modes. Thus it is possible to run them continuously.

Table 4 The multimode fault diagnosability matrix for the residuals of the dynamically configurable battery pack. The variable on := on1 ∨ on2.

	NF	f_cell1	f_i1	f_v1	f_cell2	f_i2	f_v2	f_vpack	f_ipack
f_cell1	**T**	**F**	**T**	on1	**T**	**T**	**T**	**T**	**T**
f_i1	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**T**	**T**
f_v1	**T**	on1	**T**	**F**	**T**	**T**	**T**	**T**	**T**
f_cell2	**T**	**T**	**T**	**T**	**F**	**T**	on2	**T**	**T**
f_i2	**T**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**
f_v2	**T**	**T**	**T**	**T**	on2	**T**	**F**	**T**	**T**
f_vpack	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**F**	**T**
f_ipack	on	on	on	on	on	on	on	on	**F**

Table 5 Fault signature matrix for the selected residuals for the dynamically configurable battery pack.

Residual	f_cell1	f_i1	f_v1	f_cell2	f_i2	f_v2	f_vpack	f_ipack
16	on1	on1	**F**	on2	on2	**F**	**T**	**F**
7	**F**	**F**	on1	**F**	**F**	on2	**T**	**F**
4	**F**	**T**	**F**	**F**	**F**	**F**	**F**	on1
12	**T**	**F**	**T**	**F**	**F**	**F**	**F**	on1
1	**F**	**F**	**F**	**F**	**T**	**F**	**F**	on2
3	**F**	**F**	**F**	**T**	**F**	**T**	**F**	on2

This is particularly important for the residual generators with dynamic states. If these were shut off reinitialization of the states would be a problem to be solved. In the example, the residuals influenced by f_cellj have dynamics, i.e., residuals 3, 12, and 16.

8 Computational Complexity

The algorithm is linear in the number of residuals, quadratic in the number of faults, and worst case exponential in the number of operation modes variables. To empirically evaluate the algorithm's performance, the TestSelection-algorithm is applied to modular battery packs with 2, 4, and 6 modules. The results are given in Table 6. For the example, the computation time agrees well with the theoretical complexity, except that the computation time increases linearly with the number of mode variables.

It is also interesting to note that there are solutions with fewer residuals achieving maximum diagnosability for the 4 and 6-module cases. There is a solution with 10 residuals for the 4-module case and 14 for the 6-module case. In these cases, the smaller solutions are subsets of the selected ones. It is not surprising the algorithms do not find optimal solutions since they are greedy.

Table 6 The TestSelection-algorithm applied to modular battery packs with 2, 4, and 6 modules.

Modules/mode variables	Faults	Residuals	Selected Residuals	Time
2	8	15	6	44 ms
4	14	93	14	1.6 s
6	20	747	20	37 s

9 Conclusions

Residual selection for multimode systems has been described. An algorithm is proposed for selecting a small set of residuals with the same diagnosability as all residuals. It takes a multimode version of a fault signature matrix as input. This means that any residual generation technique, model-based or data-driven, is applicable as long as a correct fault signature matrix can be defined. The user can define or use one of the proposed improvement functions to map a certain diagnosability to a performance score. Properties that could be considered in the improvement function are e.g. the locality of tests searching for a distributed solution, and the computational complexity or robustness of the residual generators. A Python package has been developed and is available at `https://github.com/MattiasKrysander/Multimode-Test-Selection`. The algorithm is demonstrated with good results on a modular and dynamically reconfigurable battery pack. Packs of different sizes are analyzed and the results show that the algorithm is efficient and can be applied to systems with many potential residuals and operation modes.

References

1 Elodie Chanthery, Anna Sztyber, Louise Travé-Massuyès, and Carlos Gustavo Pérez-Zuñiga. Process decomposition and test selection for distributed fault diagnosis. In *33th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2020)*, volume 12144, Kitakyushu, Japan, September 2020. doi:10.1007/978-3-030-55789-8_78.

2 Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2):197–222, 1992. doi:10.1016/0004-3702(92)90027-U.

3 Amir Fijany and Farrokh Vatan. A novel method for derivation of minimal set of analytical redundancy relations for system diagnosis. In *2010 IEEE Aerospace Conference*, pages 1–14, 2010. doi:10.1109/AERO.2010.5446823.

4 Ioannis Filippidis, Sofie Haesaert, Scott Livingston, and Mario Wenzel. TuLiP/dd. `https://github.com/tulip-control/dd`, 2022. Accessed: November 15th, 2023.

5 Erik Frisk and Mattias Krysander. Residual selection for consistency based diagnosis using machine learning models. *IFAC-PapersOnLine*, 51(24):139–146, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018. doi:10.1016/j.ifacol.2018.09.547.

6 Erik Frisk, Mattias Krysander, and Daniel Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293, 2017. 20th IFAC World Congress. doi:10.1016/j.ifacol.2017.08.504.

7 Erik Frisk, Mattias Krysander, and Daniel Jung. Fault diagnosis toolbox for matlab and python, 2024. Accessed: 2024-08-26. URL: `https://faultdiagnosistoolbox.github.io`.

8 J. Gertler. Structured residuals for fault isolation, disturbance decoupling and modelling error robustness. *IFAC Proceedings Volumes*, 25(4):15–23, 1992. IFAC Symposium on On-line Fault Detection and Supervision in the Chemical Process Industries, Newark, Delaware, 22-24 April. doi:10.1016/S1474-6670(17)50210-4.

9 Xuping Gu and Xianjun Shi. A review of research on diagnosability of control systems based on structural analysis. *Applied Sciences*, 13(22), 2023. doi:10.3390/app132212241.

10 Fatemeh Hashemniya, Benoît Caillaud, Erik Frisk, Mattias Krysander, and Mathias Malandain. Fault diagnosability analysis of multi-mode systems. *IFAC-PapersOnLine*, 58(4):210–215, 2024. 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024. doi:10.1016/j.ifacol.2024.07.219.

11 Daniel Jung and Christofer Sundström. A combined data-driven and model-based residual selection algorithm for fault detection and isolation. *IEEE Transactions on Control Systems Technology*, 27(2):616–630, 2019. doi:10.1109/TCST.2017.2773514.

12 Hamed Khorasgani and Gautam Biswas. Structural fault detection and isolation in hybrid systems. *IEEE Transactions on Automation Science and Engineering*, 15(4):1585–1599, 2018. doi:10.1109/TASE.2017.2749447.

13 Hamed Khorasgani, Gautam Biswas, and Daniel Jung. Structural methodologies for distributed fault detection and isolation. *Applied Sciences*, 9(7), 2019. doi:10.3390/app9071286.

14 Ramon Sarrate, Vicenc Puig, Teresa Escobet, and Albert Rosich. Optimal sensor placement for model-based fault detection and isolation. In *2007 46th IEEE Conference on Decision and Control*, pages 2584–2589, 2007. doi:10.1109/CDC.2007.4434452.

15 Michael Schmid, Emanuel Gebauer, and Christian Endisch. Structural analysis in reconfigurable battery systems for active fault diagnosis. *IEEE Transactions on Power Electronics*, 36(8):8672–8684, 2021. doi:10.1109/TPEL.2021.3049573.

16 L. Trave-Massuyes, T. Escobet, and X. Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 36(6):1146–1160, 2006. doi:10.1109/TSMCA.2006.878984.

17 L. Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27:1–16, 2014. doi:10.1016/j.engappai.2013.09.018.

Transformer-Based Signal Inference for Electrified Vehicle Powertrains

Stan Muñoz Gutiérrez
Institute of Software Technology, TU Graz, Austria

Adil Mukhtar
Institute of Software Technology, TU Graz, Austria

Franz Wotawa[1]
Institute of Software Technology, TU Graz, Austria

Abstract

The scarcity of labeled data for intelligent diagnosis of non-linear technical systems is a common problem for developing robust and reliable real-world applications. Several deep learning approaches have been developed to address this challenge, including self-supervised learning, representation learning, and transfer learning. Due largely to their powerful attention mechanisms, transformers excel at capturing long-term dependencies across multichannel and multi-modal signals in sequential data, making them suitable candidates for time series modeling. Despite their potential, studies applying transformers for diagnostic functions, especially in signal reconstruction through representation learning, remain limited. This paper aims to narrow this gap by identifying the requirements and potential of transformer self-attention mechanisms for developing auto-associative inference engines that learn exclusively from healthy behavioral data. We apply a transformer backbone for signal reconstruction using simulated data from a simplified powertrain. Feedback from these experiments, and the reviewed evidence from the literature, allows us to conclude that autoencoder and autoregressive approaches are potentiated by transformers.

2012 ACM Subject Classification Hardware → Fault models and test metrics; Computing methodologies → Machine learning algorithms; Hardware → Failure recovery, maintenance and self-repair

Keywords and phrases Signal Inference, Deep Learning, Self-Supervised Learning, Multimodal Transformer Autoencoder, Electric Vehicle, Powertrain, Electric Motor

Digital Object Identifier 10.4230/OASIcs.DX.2024.29

Category Short Paper

Supplementary Material *Software (MATLAB Notebook)*: `https://zenodo.org/records/13879822`

Funding This study was conducted within the framework of the project ARCHIMEDES, receiving funding from the Key Digital Technologies Joint Undertaking (KDT JU) - the Public-Private Partnership for research, development, and innovation under Horizon Europe – and National Authorities under Grant Agreement No 101112295 funded by the European Union and the FFG under Grant Agreement No FO999899377.

1 Introduction

Data-driven diagnosis solutions rely on characterizations of the target system's healthy, and sometimes abnormal, observed behavior. Machine learning models are usually preferred when diagnosis tasks are needed for systems exhibiting partial observability, high dimensionality, non-linear dynamics, temporal multiscale dependencies, delays, low signal-to-noise ratio, and high epistemic uncertainty. Deep learning methods excel at addressing challenging diagnosis

[1] corresponding author

© Stan Muñoz Gutiérrez, Adil Mukhtar, and Franz Wotawa;
licensed under Creative Commons License CC-BY 4.0

problems, provided sufficient data is available, and a proper methodology is followed for their training and validation. In 2017, the transformer architecture[36] marked a paradigm shift regarding the handling of sequential data in deep learning applications. Attention-based models have since become central to developing natural language processing (NLP) solutions, with transformers positioning themselves as state-of-the-art technologies. They outperform long short-term memory (LSTM) networks, and recurrent neural networks (RNNs) in various tasks, including language translation, text classification, and natural language reasoning. Their performance is primarily attributed to their attention mechanisms, which enable them to exploit intricate contextual temporal dependencies of sequential behavior.

The performance of transformer architectures extends beyond natural language processing (NLP) applications. According to the taxonomy identified by Islam et al. [14], there are 5 main transformer-based application domains: NLP, computer vision (CV), multi-modality (MM), audio/speech (A&S), and signal processing (SP)[2]. Research in the MM, A&S, and SP domains is important to diagnosis solutions because they handle digital signals as inputs rather than text. CV category is relevant too, as it is common practice to transform signals into images for deep learning applications in diagnosis. Notably, transformers are increasingly being considered for medical diagnosis based on biomedical images or physiological signals.

Our research objective is to design transformer models that enable fault detection and diagnosis using self-supervised learning. Our contributions can be summarized as:

- Identification of pre-processing steps necessary to input the transformer model.
- Identification of meta-requirements for a development framework for transformer-based intelligent diagnosis of electric drives.
- A proof of concept transformer autoencoder architecture for multimodal signal inference.
- A discussion on the suitability of these models for deployment in electrified powertrains.

2 Background and Literature Review

2.1 Learning Modalities

Machine learning (ML) techniques are increasingly being adopted in various scientific disciplines [17] and industrial applications [1]. The key characteristic of ML techniques is the ability to extract and learn *linear* and *non-linear* decision boundaries from within the observational data, i.e., feature space, of an underlying system. Hence, the objective is to approximate a learning function $F(\cdot)$ for mapping the decision, such as classification or regression, through predictor variables, called feature space $\{x_1, x_2, x_3, ..., x_n\}$. Different learning modalities exist, including *supervised* [23, 16], *unsupervised* [34], and *semi-supervised* [35, 26]. Supervised methods primarily require annotated labels for each single observation [16] in the form of input/output pairs, whereas, unsupervised methods heuristically infer the model of the unique representation(s) present in the data and do not require prior labels. Semi-supervised techniques can exploit both unlabeled and label data. In our case, we focus on *self-supervised* learning, a type of unsupervised learning that seeks to uncover the inherent structural dependencies within the data samples. Additionally, our model is based on the concept of *autoencoders*, which will be introduced next.

[2] The classification is based on a comprehensive literature review that selected 641 works. The relative number of papers shows that applications outside NLP are significant: NLP (40.0%, 257 papers), CV (30.7%, 197 papers), MM (14.7%, 94 papers), A&S (10.9%, 70 papers), and SP (3.6%, 23 papers). Some examples of applications outside the NLP domain include image detection, segmentation and classification, visual automatic captions, speech recognition, and medical signal processing.

2.2 Autoencoders

In 1993, Hinton and Zemel [13] understood that unsupervised learning algorithms can be seen as variants of PCA or vector quantization and that the *autoencoder* model (AE) encompasses both. In a recent literature review, Yang et al. [39] presents the foundations of *representational learning* from the perspective of the autoencoder family, covering their contribution to the field of intelligent fault diagnosis. The architecture of autoencoders entails dimensionality reduction, which aligns well with Hinton and Zemel's theoretical formulation that interprets them as solving a minimum-length description problem. When implemented using deep learning models, they are conceptually divided into two components: the *encoder* and the *decoder*. The encoder is typically depicted as a funnel-like structure that progressively projects a high-dimensional input into a lower-dimensional space, also referred to as the *latent space*; the decoder takes this representation and projects it back to the original input space, a process referred to as *reconstruction*. Learning algorithms for the parameters of such structures are driven primarily by a loss function designed to minimize the reconstruction error. A well-accepted taxonomy of AEs categorizes them into (1) denoising, (2) sparse, (3) contractive, and (4) variational. We will only introduce denoising autoencoders here, the reader can find more about the other categories in [27, 15, 39].

In [37], Vincent et al. pose the question of what constitutes a good representation. In answering this question, they come out with desiderata that we rephrase as follows: (1) preserve as much information as possible about the input, (2) embedding in a *pre-defined* representational space, (3) invariance to partial destruction of the input, and (4) recall from partial observable inputs. Therefore, denoising autoencoders (DAEs) are nuanced and explicit about requiring *robust representations* and capturing the internal structural associations needed to solve the *fill-in-the-blanks* problem. The learning in DAEs is based on a simple but powerful principle, *the autoencoder should repair partially corrupted inputs*.

2.3 Transformers for anomaly detection in time series

In [29], Shin et al. obtained state-of-the-art anomaly detection performance in time series using *AnoFormer*, a pure transformer-based generative adversarial network framework. Self-supervised training was performed using a two-stage masking strategy. In the first stage, masks were randomly obtained from a predefined pool of masks, then in the second stage, an entropy-based resampling strategy was applied. In their framework, Shin et al. work directly in the time domain, building a set of tokens from a quantized version of the normalized signal. A matrix-based linear embedding is automatically learned, followed by standard sinusoidal positional encoding. The training was adversarial, using only normal behavior data. The generator and the discriminator (critic) were implemented using transformer encoders. Once the generator is trained, the decision about normal or abnormal behavior is based on the generator's reconstruction error of the input signal. AnoFormer outperformed BeatGAN [43] (adversarial reconstruction-based CNN model); TadGAN, RAE-Ensemble, and RAMED (RNN-based models); as well as Anomaly Transformer (transformer-based) for the four datasets in the benchmark when evaluated using AUROC, AUPRC, and F1-score.

2.4 Attention-based diagnostic models for electric motors

In [18], Li et al. analyze the performance of a deep learning architecture enhanced with an attention mechanism when diagnosing bearing faults. They apply an attention-as-weighing approach, where a positive weight captures the relative importance of each input segment. High-level representations of the input are computed by a deep learning architecture that

stacks convolutional layers followed by a long short-term memory (LSTM) model with a fully connected layer at the output end. Attention weights are computed by feedforward single-layer applied to the high-level representations. Finally, the high-level representations of the segments are linearly combined using the attention weights generating the inputs used for the classification stage that is implemented with a softmax regression layer. They analyzed three different input formats, (1) raw vibration signal, (2) envelope spectrum, and (3) frequency spectrum, and benchmarked their architecture against traditional deep neural network architectures with and without their attention mechanism in place. Attention mechanisms enhanced diagnosis performance across input formats and model variants.

Transformer-based diagnosis of mechanical faults has been demonstrated by Ding et al. [10]. Their study engineered a transformer architecture for rolling bearing diagnosis that selects as inputs the synchrosqueezed wavelet transforms (SWT) of vibration signals to capture time-frequency information that is then split into sequential fragments. Using each sequential fragment as input, a tokenization process was applied to obtain high-dimensional embeddings of the time-frequency fragments by applying learnable linear mappings and considering 1-D and 2-D learnable positional encodings. After the tokenization stage, a transformer encoder is trained to predict the next token, treating token characterization as an *autoregressive prediction problem*. Finally, the fault diagnosis is performed by a multilayer perceptron using one-hot encoding and a softmax layer at the output. Ding et al. benchmarked their model against a multilayer perceptron, a CNN, and a gated recurrent unit neural network (GRU). Their experiments show that their model outperforms the others across different operating conditions and signal-to-noise ratios.

3 Diagnosis of EVPs

3.1 Fault modes in motor systems

In [6], Chen et al. present a taxonomy of common fault types occurring in permanent magnet synchronous motors (PMSMs), which are the most prevalent type of motor used in powertrains of electrified road vehicles[41]. Faults are classified as electrical, mechanical, and magnetic. Faults obey "mutual catalytic" relationships among them, meaning that manifested faults are likely to trigger dependent faults. These dependent faults rapidly intensify each other, compromising the safety and integrity of the system and shortening its remaining useful life. Because of this, we are interested in the early detection of manifested faults also known as *incipient* faults, whose detection and diagnosis are difficult due to their low energy spread-out frequency components [18].

Ullah and Hur [30] review the detection and identification of inter-turn short (-circuit) faults (ITSFs) and irreversible demagnetization faults (IDFs) occurring in permanent magnet (PM) type machines. ITSFs are caused by insulation damage in the winding of PM motors causing a short circuit in the same phase of the machine [3]. Meanwhile, IDFs are due to the nonlinear high sensitivity of PMs to their operating temperature, overload conditions, mechanical stressors, reverse magnetic fields resulting from an ITSFs, and aging. Ullah and Hur identity fault indices based on: (1) stator current (SC), (2) stator voltage (SV), (3) parameter estimation (PE), (4) magnetic flux (MF), and (5) mechanical output (MO).

[3] ITSFs are one type of winding insulation faults, other types include coil-to-coil (CCFs), phase-to-phase (PPSFs), and phase-to-ground (PGSFs) [9].

3.2 Sensors

Prognosis and health monitoring (PHM) applications often require sensors that monitor modalities independent of the system's primary functional requirements. For instance, consider the use of thermal sensors in various powertrain components. These sensors are safety-critical and provide key parameters for models of their *physics of degradation*, thus contributing to an accurate estimation of the components' remaining useful life (RUL). In the case of Electric Vehicle Powertrains (EVPs), a combination of embedded and networked sensors should be selected, with some networked sensors capable of self-diagnosis.

Diagnosing sensor faults effectively is hard because they manifest in several ways, including bias sensor fault (BiasSF) [40, 42], cyclic sensor fault (CycSF) [40], erratic sensor fault (ErrSF) [40], gain fault on sensor (GFOS) [28], sensor abrupt fault (SAF) [40], sensor omission fault [3], stuck sensor fault (StckSF) [40], and spike sensor fault (SpkSF) [40].

3.3 Signal modalities

3.3.1 Electric

SV signals are used extensively in fault-tolerant control. It relies on measuring the voltage at the neutral point of the stator windings and the DC midpoint of the inverter. The zero-sequence-voltage components (ZSVCs) are informative of anomalies because they are unaffected by the drive on healthy motors, yet, they are highly coupled with the speed when an ITSF manifests. In particular, the first harmonic that can be efficiently computed by the Vold-Kalman filtering order tracking (VKF-OT) algorithm is suitable to be used as the basis for ITSFs indicators [33, 32]. SC signals are the inputs to many algorithms for fault detection of ITSFs and are the concern of the machine current signature analysis (MCSA) field. MCSA uses different spectral characterizations based on Hilbert-Huang, Wavelet, or Fourier transformations. Ullah and Hur [30], report on several AI techniques that take SC inputs and have shown to be successful in implementing fault detection, diagnosis, and severity assessment of ITSFs: [22], [24], [25]. In particular, [25] and [22] were designed to assess the severity of faults using a particle swarm optimization (PSO) algorithm, the former, and a multilayer perception (MLP), the latter. Both [25] and [24] can disambiguate ITSFs from load fluctuations across different operating conditions.

3.3.2 Vibration

In [44], Zou et al. diagnoses ball faults (BallFs), inner raceway faults (IRFs), and outer raceway faults (ORFs) from vibration signals of gearboxes. They treat vibration signals with Ensemble empirical mode decomposition (EEMD). EEMD conducts an ensemble of experimental trials, where independent Gaussian noise signals are added to the observed vibration signal. After decomposing each one of the signals of the ensemble using empirical mode decomposition (EMD) into the intrinsic mode functions IMFs, the method combines the signals within each one of the IMFs, the effect is that the Gaussian signals cancel each other out, but also increase the signal-to-noise ratio. EEMD. They apply a *kurtosis filter*, consisting of an IMF selection mechanism that selects the k=6 IMFs that are most informative for the faults. An LSTM uses the 6 IMFs and can achieve accuracies of 99.98%. In [20], Li uses images with spectral information of vibration signals to diagnose rotor unbalance (RU), shaft misalignment (SM), BallFs, IRFs, and ORFs. Variational mode decomposition (VMD) was applied to the vibration signal. Then the intrinsic mode functions (IMFs) components were transformed using the Hilbert transformation. From these transformed components, the

Hilbert marginal spectrum was computed. Finally, this was transformed into an image that was the input to three different deep neural network models: ResNet101, GoogLeNet, and AlexNet. ResNet101 obtained the highest diagnostic accuracy (94%).

3.3.3 Acoustic

In the work of Choudhary et al. [8], CNN models have been trained to diagnose ORFs, IRFs, and BallFs using exclusively acoustic signals. Using *transfer learning* target also diagnosis of BRBFs, rotor misalignments (RMAs), and BFs. The accuracy in the target domain is close to that of the source domain (95.8% and 94.98% no load, 94.2% and 97.68% with load). In [12], Glowacz study diagnosis and severity assessment of faulty ring of squirrel-cage (FRSCs), and BRBFs (1 or to broken bars) using acoustic signals. He compares the performance of a nearest neighbor classifier (NNC), a MLP, and a word-code-based classification (WCBC). MLP outperformed the other approaches, exhibiting perfect accuracy.

3.4 Multi-signal

Multi-signals, meaning signals from the same modality monitoring subsystems with coupled dynamics, are informative of sensor faults. In [28], a current sensor fault in an inverter was diagnosed by estimating the gain from the three-phase currents, reference torque, and motor speed. After a feature selection step, the signals from all current sensors were used to diagnose the GFOS for the current sensor of a single phase of the inverter. In [7], Chen and Li performed sensor fusion from vibration signals, simultaneously acquired from three accelerometers monitoring a motor system. The inputs to the model are 18 indicators computed in the time and frequency domains. The model comprises a bank of 18 sparse autoencoders (SAEs). These SAEs compute a *sparse overcomplete representation* that is then fused into a single channel. The fused feature signals are the inputs of a deep belief network DBN that diagnoses IRFs and ORFs and assesses their severities.

3.5 Multimodal

Liang et al. [19], used single-phase currents and vibration signals to diagnose ITSFs. By applying sparse linear decomposition (SLD) and the algorithm orthogonal matching pursuit (OMP), they computed a representation from which a feature vector was obtained. An SVM was then trained to detect ITSFs in PMSMs. The SVM achieves perfect accuracy on a small testing dataset of 6 feature vectors (3 healthy, and 3 abnormal). Ullah et al. [31] integrate vibration and current signals to diagnose IDFs and BFs of PMSMs using a CNN (VGG-16 architecture). They convert both the FFT spectrum of current signals and the temporal vibration signals to 64×64 2D images with 3 channels (one for each modality and the third zero padded). Using this setting, they obtain 96.56% accuracy with real data from an experimental platform. In [21], Ma et al. integrate vibration and acoustic information using a deep coupling autoencoder trained with a loss function that uses information-theoretic measures of signal correlations. Their model diagnoses gearboxes (GB) defects and BFs, outperforming single modality and a variant based on simple signal stacking.

3.6 Input format

The input format of signals refers to the domain of representation. Some of the most widely used include the fast Fourier transform (FFT), wavelet transform (WT), wavelet packet decomposition (WPD), short-time Fourier transform, empirical mode decomposition (EMD),

Hilbert transform (HT) and Hilbert-Huang transform (EMD + HT). Transient vibrations produce signals that are not stationary, the choice of representation can enhance or hinter conveying this information to the diagnostic model. We refer the reader to [11] for a detailed discussion and analysis of several of the most important input formats.

4 The signal inference transformer

Our signal inference transformer is part of the research supported by the project ARCHIMEDES [2], for the acceleration of the energy transition, safety, and security for the future society 5.0.

4.1 Metarequirements

The high-level requirements for the signal inference transformer can be summarized in the following list:

1. The deep learning model should be trained exclusively on data representative of the healthy behavior of the system learned in self-supervised learning mode.
2. Multimodal support for signals relevant to automotive powertrains of electrified vehicles.
3. Inference of signals should support the prediction of future scenarios given projected control signals and operational state.
4. Inference of whole signal channels (virtual sensors).
5. Explanations (signal reconstruction) of arbitrary channels.
6. Knowledge injection by simulation models that enable parameter estimation (relevant for fault detection and prognosis).
7. Knowledge injection for intelligent masking by definition of causal constraints.
8. A flexible open-source processing pipeline should be defined and implemented to automate as much as possible the development of the model.
9. Objective metrics should be automatically computed and human-legible reports in the spirit and facilitating quality assurance.
10. The framework facilitates downstream adaptation to implement several diagnostic functions via semi-supervised learning.

4.2 Conceptual solution

A conceptual diagram of the transformer-based signal inference solution is shown in Figure 1. The transformer is fed with a sequence that integrates the sequential presentation of multimodal sequential tokens augmented with control tokens. A key aspect of future research is the appropriate tokenization for the signals relevant to the diagnostic functions of electrified vehicle powertrains. The controller is a module that parameterizes the sequencer handling query tokens and should not be confused with the vehicle's controller signals[4].

5 Case Study

A model of a simplified powertrain [38] was used to generate simulation data with meaningful signals that could be used to prototype ideas and develop our processing pipeline and solution. The signals shown in Figure 1 are part of the simulation dataset.

[4] The vehicle's control signals are encoded as multimodal channels.

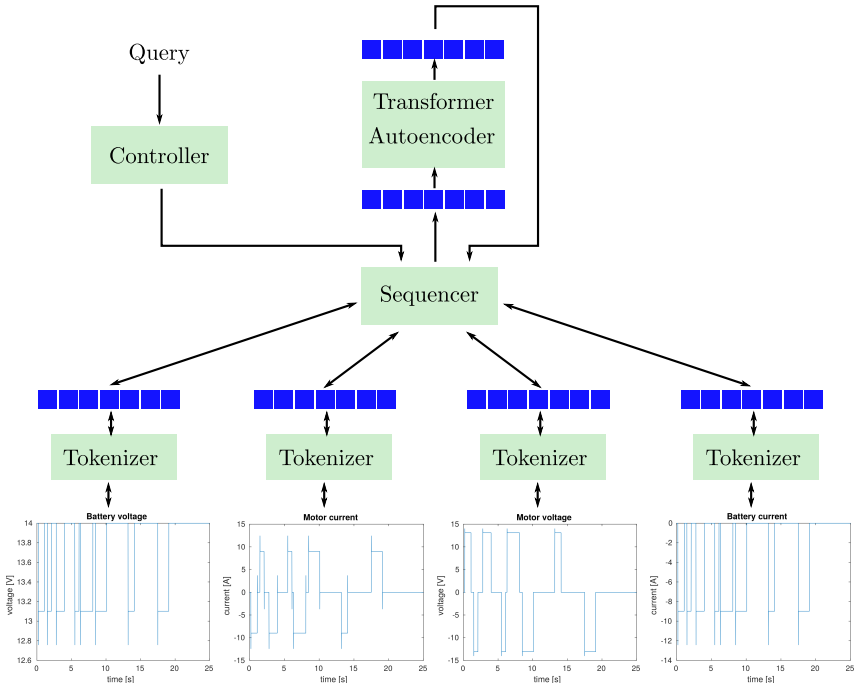


Figure 1 Transformer-based signal inference box.

5.1 Transformer architecture

We evaluated the definition of an architecture in the platform MATLAB. Figure 2 shows one of the several examples of architectures we are exploring. These architectures comprise an autoencoder implemented with a transformer encoder backbone. For anomaly and fault-detection applications, it is often sufficient to have this type of architecture, therefore we are starting there. The number of modules in the transformer encoder is a customizable parameter. We have found encouraging results, even with naive representations for the tokens, such as representing masks with negative constants.

5.2 Self-supervised approach

The training of the autoencoder follows the DAE methodology, although we don't want to refer only to masking as the corruption of information for the transformer to denoise but would like to have a richer set of control tokens in place. We found encouraging applications in the any-to-any models that are capable of inference across several modalities[4]. We are considering pre-training schemes to assist with the transformer encoder before plugging it into the decoder (see Figure 1 in Bao et al. [5]).

5.2.1 The processing pipeline

In Figure 3, we illustrate our preprocessing pipeline with an example. The computer simulation model signals were acquired at 10 kHz. The first preprocessing step consists of scaling and downsampling the signals, in the future model, several temporal scales need to be accounted for. The next step splits the signal into windows (10 ms length in the example), that can potentially overlap (50% in the example). The input sequence is obtained from the window by stacking and interleaving samples so that synchronous samples remain

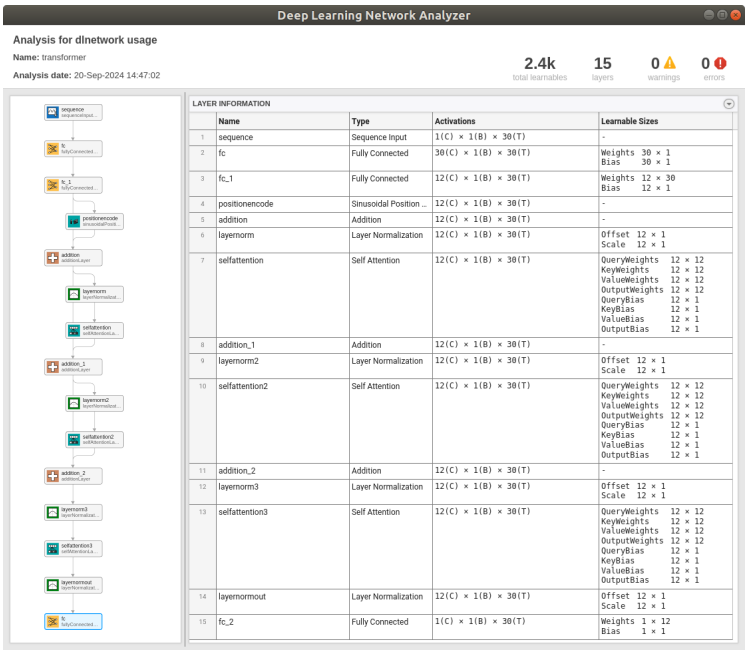


Figure 2 A potential transformer encoder architecture.

close to each other. Note that we do not use multiple channels for different signals because that would treat them as the same token, and we would lose resolution in the queries. For simulation data, the system stays in steady-states most of the time. Using raw window data would result in highly imbalanced datasets. To overcome this, we run a *uniqueness filter* (UF) with a tolerance factor (0.001 for the example). After this step, for the 25 s run shown in the plots, the 4999 windows shown in subfigure (4) became only 48 (subfigure (5)). It is interesting to note that by overlaying the windows in the original signals, the windows happen to capture the signal transitions only (subfigure (A)).

5.3 Datasets generation

To avoid *data leakage*, we don't partition the training, validation, and testing sets at the window level because each central window overlaps with its two neighbors. Knowing that the unique windows are scarce, we instead computed *segments of contiguous windows* (SCWs). In the example, there are 15 such segments (one segment can be seen in Figure 3) (B). We define training, validation, and test ratios as 0.6, 0.2, and 0.2 respectively. After random sampling without replacement, in the example, 9 segments were allocated for training, 3 for validation, and 3 for testing.

5.4 Self-supervised learning by masking

We trained the system by presenting random masked versions of the training samples at the input and the corresponding complete (unmasked) sequence at the output. The loss function is the mean squared (reconstruction) error (MSE).

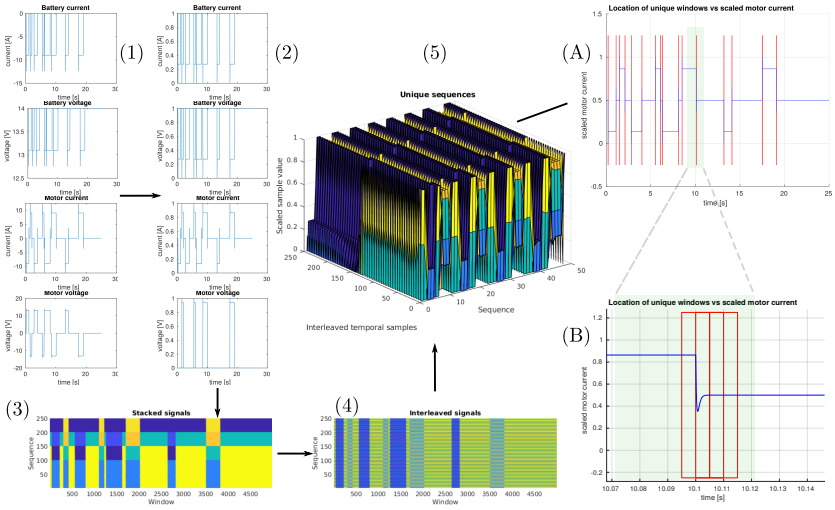


Figure 3 Preprocessing pipeline: (1) raw signals (h-bridge signal is omitted), (2) downsampling and scaling, (3) window extraction and stacking, (4) signal interleaving, (5) unique windows; (A) overlay of the unique windows on the motor current, and (B) zoomed region showing a segment of three overlapped consecutive windows.

5.5 Preliminary results

Figure 4 shows a histogram of reconstruction errors for the test dataset, after 196 epochs of training (MSE: training 0.0034, validation 0.0042, testing 0.0091). We provide an open-source interactive notebook in supplementary material with all parameters used in the example.

6 Discussion and future work

The progressive refinement of the AE concept from an information-theoretic perspective to that of a deep self-associative machine is opening new avenues for addressing challenging problems in PHM. Evidence from the literature supports the notion that integrating multimodal information enhances diagnostic performance. Attending to that conclusion, we have added multimodality to our desiderata. Although we are not yet in a position to present concrete results in this direction, the research community has shown transformers to be able to handle tens of modalities [4]. *Tokenization* is key to this aim, as provides a *unified representation* digestible by a transformer architecture. Predicting future scenarios is becoming relevant in the context of driving automation and eco-driving. The functional requirements in our desiderata are ambitious but follow naturally from achieving accurate learning of deep structural dependencies in the signals.

Although a simple idea, learning by filling in the blanks requires a deeper understanding of how to properly dose and structure those blanks. An advantage of diagnosing technical systems is that our machines have stringent specifications, and are designed and evaluated by mathematical and computational models. We added *Knowledge injection by simulation models* to our requirements because it introduces interesting possibilities. For the transformer architecture, it does not matter if the signal is learned from synthetic data or real-world data. Simulated data can be fed into the system during training and serve as a baseline for real-world data, while also addressing scenarios for rare events that might never be observed in actual data. Moreover, we understand *qualitatively* the dependencies among subcomponents and to some degree of the signals. This is an advantage as compared with

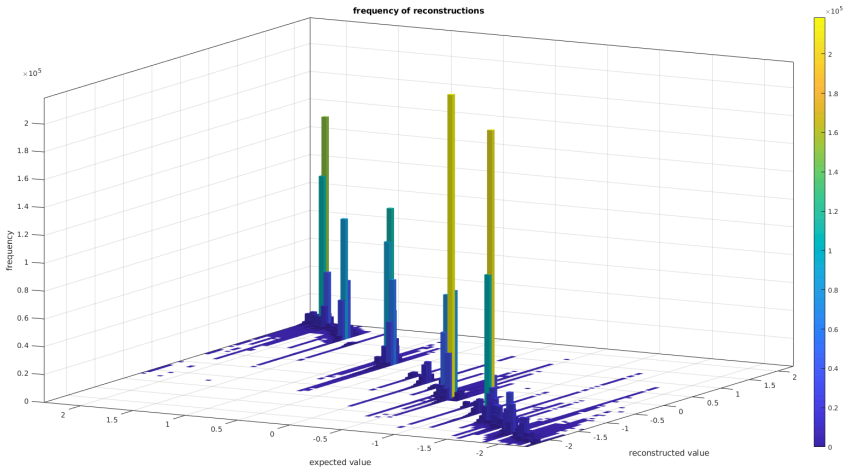


Figure 4 Preliminary results: (a) Histogram of reconstruction errors for the test dataset.

other fields because masking does not need to try so many combinations but a few *engineered* causal links. In other words, we don't need to start from *tabula rasa*, by incorporating engineering knowledge and constraints, the transformer autoencoder can be made more reliable by eliminating spurious dependencies not distinguishable from the data alone.

Having finite dictionaries aligns well with the problem of handling big data in modern vehicles, compacting the requirements and bandwidths. The challenge consists of optimizing the tokenization process while satisfying the desiderata of Vincent et al. [37]. Compressing is concomitant to reducing the energy footprint, another good aspect of these approaches. Digital twins in the context of Industry 4.0 and Society 5.0 would greatly benefit from standardized dictionaries, task-agnostic pre-trained and optimized embeddings, and open pre-trained foundational transformer models.

An important consideration for the application of transformer-based signal inference is the computational cost and energy demands associated with these technologies. It remains unclear whether current advancements are mature and efficient enough to be viable and environmentally friendly for onboard implementation in electric vehicles. Although numerous studies are exploring the feasibility of bringing transformers to edge computing for IoT, the timeline for these approaches to become mainstream remains uncertain. Nevertheless, we recognize other promising use cases where signal inference systems could offer significant value. For instance, simulations are integral to the qualification process in automotive development. The potential of these technologies to create mock digital twins could greatly enhance evaluation procedures by incorporating generative capabilities. This is particularly relevant given the proven effectiveness of variational autoencoders in other domains, which could be adapted to generate subsystem behaviors during development, thereby strengthening the overall quality assurance process.

7 Final remarks

For centuries, we have accumulated extensive knowledge in electric drive engineering. This knowledge, encapsulated in mathematical and computational models, enables us to make remarkably accurate predictions of the expected healthy behavior of electric drives across a wide range of conditions within their specified operational envelope.

While the physics of failure is well understood for most common faults, emerging technologies challenge our grasp of the highly coupled, nonlinear multiphysics resulting from the deep integration of electrical and mechanical components in electric vehicle powertrains.

A wide array of classical and data-driven techniques exists for detecting and diagnosing manifested faults in these systems. However, a significant gap remains in incorporating prognostics, partly due to the limited understanding and validation of theoretical models of hybrid high-power electronics degradation and the accumulated aging of motor components as a function of their operational history across varying environmental conditions.

Although transformer architectures are relatively new to the deep learning paradigm, they hold promise for unifying existing knowledge captured by computational models with real vehicle operational data to deliver the adaptability and learning required for the condition monitoring and prognosis of highly automated vehicles. Our conceptual multimodal transformer autoencoder seeks to introduce a foundational technology that can catalyze further advancements through its high adaptability while maintaining safety and quality standards in electrified vehicle powertrains.

References

1 Sheena Angra and Sachin Ahuja. Machine learning and its applications: A review. In *2017 international conference on big data analytics and computational intelligence (ICBDAC)*, pages 57–60. IEEE, 2017.

2 ARCHIMEDES Consortium. ARCHIMEDES ensures lifetime with digital means. `https://www.archimedesproject.eu/`, 2024. Accessed on September 9, 2024.

3 Ritik Argawal, Dattatraya Kalel, M Harshit, Arun D Domnic, and R Raja Singh. Sensor Fault Detection using Machine Learning Technique for Automobile Drive Applications. In *2021 National Power Electronics Conference (NPEC)*, pages 1–6, 2021. `doi:10.1109/NPEC52100.2021.9672546`.

4 Roman Bachmann, Oğuzhan Fatih Kar, David Mizrahi, Ali Garjani, Mingfei Gao, David Griffiths, Jiaming Hu, Afshin Dehghan, and Amir Zamir. 4M-21: An Any-to-Any Vision Model for Tens of Tasks and Modalities, 2024. `doi:10.48550/arXiv.2406.09406`.

5 Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers, 2022. `doi:10.48550/arXiv.2106.08254`.

6 Yong Chen, Siyuan Liang, Wanfu Li, Hong Liang, and Chengdong Wang. Faults and Diagnosis Methods of Permanent Magnet Synchronous Motors: A Review. *Applied Sciences*, 9(10):2116, May 2019. `doi:10.3390/app9102116`.

7 Zhuyun Chen and Weihua Li. Multisensor Feature Fusion for Bearing Fault Diagnosis Using Sparse Autoencoder and Deep Belief Network. *IEEE Transactions on Instrumentation and Measurement*, 66(7):1693–1702, 2017. `doi:10.1109/TIM.2017.2669947`.

8 Anurag Choudhary, Tauheed Mian, Shahab Fatima, and B K Panigrahi. Deep Transfer Learning Based Fault Diagnosis of Electric Vehicle Motor. In *2022 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, pages 1–6. IEEE, December 2022. `doi:10.1109/PEDES56012.2022.10080274`.

9 Yao Da, Xiaodong Shi, and Mahesh Krishnamurthy. A New Approach to Fault Diagnostics for Permanent Magnet Synchronous Machines Using Electromagnetic Signature Analysis. *IEEE Transactions on Power Electronics*, 28(8):4104–4112, 2013. `doi:10.1109/TPEL.2012.2227808`.

10 Yifei Ding, Minping Jia, Qiuhua Miao, and Yudong Cao. A novel time–frequency Transformer based on self–attention mechanism and its application in fault diagnosis of rolling bearings. *Mechanical Systems and Signal Processing*, 168:108616, 2022. `doi:10.1016/j.ymssp.2021.108616`.

11 Robert X Gao and Ruqiang Yan. Non-stationary signal processing for bearing health monitoring. *International journal of manufacturing research*, 1(1):18–40, 2006. `doi:10.1504/IJMR.2006.010701`.

12 Adam Glowacz. Acoustic based fault diagnosis of three-phase induction motor. *Applied Acoustics*, 137:82–89, 2018. doi:10.1016/j.apacoust.2018.03.010.

13 Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

14 Saidul Islam, Hanae Elmekki, Ahmed Elsebai, Jamal Bentahar, Nagat Drawel, Gaith Rjoub, and Witold Pedrycz. A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, 241:122666, 2024. doi:10.1016/j.eswa.2023.122666.

15 Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

16 Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=6950.

17 Yogesh Kumar, Komalpreet Kaur, and Gurpreet Singh. Machine learning aspects and its applications towards different research areas. In *2020 International conference on computation, automation and knowledge management (ICCAKM)*, pages 150–156. IEEE, 2020.

18 Xiang Li, Wei Zhang, and Qian Ding. Understanding and improving deep learning-based rolling bearing fault diagnosis with attention mechanism. *Signal Processing*, 161:136–154, 2019. doi:10.1016/j.sigpro.2019.03.019.

19 Siyuan Liang, Yong Chen, Hong Liang, and Xu Li. Sparse Representation and SVM Diagnosis Method Inter-Turn Short-Circuit Fault in PMSM. *Applied Sciences*, 9(2):224, January 2019. doi:10.3390/app9020224.

20 Shih-Lin Lin. Application Combining VMD and ResNet101 in Intelligent Diagnosis of Motor Faults. *Sensors (Basel, Switzerland)*, 21(18):6065, 2021. doi:10.3390/S21186065.

21 Meng Ma, Chuang Sun, and Xuefeng Chen. Deep Coupling Autoencoder for Fault Diagnosis With Multimodal Sensory Data. *IEEE Transactions on Industrial Informatics*, 14(3):1137–1145, 2018. doi:10.1109/TII.2018.2793246.

22 S.S. Moosavi, A. Djerdir, Y. Ait-Amirat, and D.A. Khaburi. ANN based fault diagnosis of permanent magnet synchronous motor under stator winding shorted turn. *Electric Power Systems Research*, 125:67–82, 2015. doi:10.1016/j.epsr.2015.03.024.

23 Vladimir Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4(51-62):56, 2017.

24 Yaw Nyanteh, Chris Edrington, Sanjeev Srivastava, and David Cartes. Application of artificial intelligence to real-time fault detection in permanent-magnet synchronous machines. *IEEE Transactions on Industry Applications*, 49(3):1205–1214, 2013. doi:10.1109/TIA.2013.2253081.

25 Yaw D. Nyanteh, Sanjeev K. Srivastava, Chris S. Edrington, and David A. Cartes. Application of artificial intelligence to stator winding fault diagnosis in permanent magnet synchronous machines. *Electric Power Systems Research*, 103:201–213, 2013. doi:10.1016/j.epsr.2013.05.018.

26 YCAP Reddy, P Viswanath, and B Eswara Reddy. Semi-supervised learning: A brief review. *Int. J. Eng. Technol*, 7(1.8):81, 2018.

27 Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 833–840, Madison, WI, USA, 2011. Omnipress. URL: https://icml.cc/2011/papers/455_icmlpaper.pdf.

28 Tunan Shen, Yuping Chen, Christian Thulfaut, and Hans-Christian Reuss. A Data Based Diagnostic Method for Current Sensor Fault in Permanent Magnet Synchronous Motors (PMSM). In *IECON Proceedings (Industrial Electronics Conference)*, volume 2019-Octob, pages 5979–5985, 2019. doi:10.1109/IECON.2019.8927667.

29 Ah-Hyung Shin, Seong Tae Kim, and Gyeong-Moon Park. Time Series Anomaly Detection Using Transformer-Based GAN With Two-Step Masking. *IEEE Access*, 11:74035–74047, 2023. doi:10.1109/ACCESS.2023.3289921.

30 Zia Ullah and Jin Hur. A Comprehensive Review of Winding Short Circuit Fault and Irreversible Demagnetization Fault Detection in PM Type Machines. *Energies*, 11(12):3309, November 2018. doi:10.3390/en11123309.

31 Zia Ullah, Bilal Ahmad Lodhi, and Jin Hur. Detection and Identification of Demagnetization and Bearing Faults in PMSM Using Transfer Learning-Based VGG. *Energies (Basel)*, 13(15):3834, 2020.

32 Julio-César Urresty, Jordi-Roger Riba, and Luis Romeral. Diagnosis of interturn faults in pmsms operating under nonstationary conditions by applying order tracking filtering. *IEEE Transactions on Power Electronics*, 28(1):507–515, 2013. doi:10.1109/TPEL.2012.2198077.

33 Julio-César Urresty, Jordi-Roger Riba, and Luís Romeral. Application of the zero-sequence voltage component to detect stator winding inter-turn faults in PMSMs. *Electric Power Systems Research*, 89:38–44, 2012. doi:10.1016/j.epsr.2012.02.012.

34 Muhammad Usama, Junaid Qadir, Aunn Raza, Hunain Arif, Kok-Lim Alvin Yau, Yehia Elkhatib, Amir Hussain, and Ala Al-Fuqaha. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE access*, 7:65579–65615, 2019. doi:10.1109/ACCESS.2019.2916648.

35 Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020. doi:10.1007/S10994-019-05855-6.

36 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. arXiv:1706.03762, doi:10.48550/arXiv.1706.03762.

37 Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1390156.1390294.

38 Franz Wotawa, David Kaufmann, Adil Mukhtar, Iulia Nica, Florian Klück, Hermann Felbinger, Petr Blaha, Matus Kozovsky, Zdenek Havranek, and Martin Dosedel. Real-Time Predictive Maintenance - Model-Based, Simulation-Based and Machine Learning Based Diagnosis. In *Artificial Intelligence for Digitising Industry – Applications*. River Publishers, 2011.

39 Zheng Yang, Binbin Xu, Wei Luo, and Fei Chen. Autoencoder-based representation learning and its application in intelligent fault diagnosis: A review. *Measurement*, 189:110460, 2022. doi:10.1016/j.measurement.2021.110460.

40 Hongkun Zhang and Wenjun Li. A New Method of Sensor Fault Diagnosis Based on a Wavelet Packet Neural Network for Hybrid Electric Vehicles. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1143–1147, 2016. doi:10.1109/CISP-BMEI.2016.7852886.

41 Jiyu Zhang, Giorgio Rizzoni, and Qadeer Ahmed. Fault Modelling for Hierarchical Fault Diagnosis and Prognosis. In The American Society of Mechanical Engineers, editor, *Dynamic Systems and Control Conference*, volume 2, October 2013. doi:10.1115/DSCC2013-3825.

42 Chunheng Zhao, Yi Li, Matthew Wessner, Chinmay Rathod, and Pierluigi Pisu. Support-Vector Machine Approach for Robust Fault Diagnosis of Electric Vehicle Permanent Magnet Synchronous Motor. In *Annual Conference of the PHM Society*, volume 12(1), page 10, November 2020. doi:10.36001/phmconf.2020.v12i1.1291.

43 Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, and Jing Ye. BeatGAN: Anomalous Rhythm Detection using Adversarially Generated Time Series. In *IJCAI*, volume 2019, pages 4433–4439, 2019. doi:10.24963/IJCAI.2019/616.

44 Ping Zou, Baocun Hou, Lei Jiang, and Zhenji Zhang. Bearing Fault Diagnosis Method Based on EEMD and LSTM. *International Journal of Computers, Communications and Control*, 15(1), February 2020. doi:10.15837/IJCCC.2020.1.3780.

Usability of Symbolic Regression for Hybrid System Identification – System Classes and Parameters

Swantje Plambeck
Institute of Embedded Systems, Hamburg University of Technology, Germany

Maximilian Schmidt
Institute of Embedded Systems, Hamburg University of Technology, Germany

Audine Subias
LAAS-CNRS, Université de Toulouse, INSA, France

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, France

Goerschwin Fey
Institute of Embedded Systems, Hamburg University of Technology, Germany

Abstract

Hybrid systems, which combine both continuous and discrete behavior, are used in many fields, including robotics, biological systems, and control systems. However, due to their complexity, finding an accurate model is a challenge. This paper discusses the usage of symbolic regression to learn hybrid systems from data and specifically analyses learning parameters for a recent algorithm. Symbolic regression is a powerful tool that can automatically discover accurate and interpretable mathematical models in the form of symbolic expressions.

Models generated by symbolic regression are a valuable tool for system identification and diagnosis, e.g., to predict future system behavior or detect anomalies. A major opportunity of our approach is the ability to detect transitions between different continuous behaviors of a system directly based on the dynamics. From a diagnosis perspective, this can advantageously be used to detect the system entering fault modes and identify their models. This paper presents a parameter study for a symbolic regression based identification algorithm.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computing methodologies → Symbolic and algebraic algorithms; Computing methodologies → Learning paradigms; Computing methodologies → Modeling methodologies

Keywords and phrases Hybrid Systems, Symbolic Regression, System Identification

Digital Object Identifier 10.4230/OASIcs.DX.2024.30

Category Short Paper

Supplementary Material
Software (Source Code): `https://github.com/TUHH-IES/SymbolicRegression4HA` [21], archived at `swh:1:dir:ee3af8e7fcea2b4be54dffa349ab4c87cdb15759`

Funding This work was supported by a fellowship of the German Academic Exchange Service (DAAD) and the ECIU Universities. The research is partially funded by the BMBF project AGenC no. 01IS22047A. It is also supported by ANITI through the French "Investing for the Future – P3IA" program under the Grant agreement n°ANR-19-P3IA-0004.

Acknowledgements Furthermore, we would like to thank Nicola Zaupa and Luca Zaccarian from LAAS CNRS for their support and comments on the power converter example.



© Swantje Plambeck, Maximilian Schmidt, Audine Subias, Louise Travé-Massuyès, and Goerschwin Fey;
licensed under Creative Commons License CC-BY 4.0



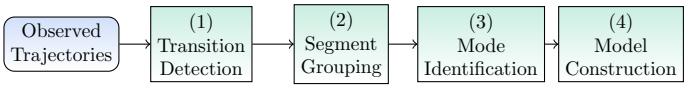


Figure 1 Steps of Hybrid System Identification [22].

1 Introduction

Hybrid systems are abstract models of systems that exhibit both continuous and discrete behavior. For this, hybrid systems have a finite number of modes, each representing a specific dynamic behavior of the system. They are used to model a wide range of systems, including cyber-physical systems or manufacturing systems, and systems with normal and faulty modes [2]. Due to their inherent combination of continuous and discrete behavior, the identification of hybrid systems is a challenging task. Nevertheless, accurate abstract models are essential for verification, diagnosis, and debugging of these systems.

We recently proposed a novel approach for automatic identification of hybrid system modes from data using Symbolic Regression (SR) [23]. Like most methods for hybrid system identification, we use a general procedure consisting of four steps as shown in Fig. 1. We use SR for the steps (1) to (3) of the identification process. In step (1), we detect transitions between different modes of a hybrid system. Step (2) and (3) are combined in one algorithm. SR is able to identify complex behavior from data [26, 15].Here, the particular opportunity of SR is the ability to detect transitions between different continuous behavior of a system directly based on the dynamics. This goes beyond existing identification strategies, which use similarities of observations to separate and group different modes of a hybrid system in observed data [3, 29, 19]. The goal of this paper is to gain a deeper insight into the capabilities and challenges of using SR for hybrid system identification, leveraging new possibilities for diagnosis on symbolic models in the future.

Like other learning algorithms for system identification, hybrid system identification with SR requires selecting parameters for learning. In this paper, we analyze the impact of the parameters on the identification of hybrid systems and discuss the trade-offs between runtime, accuracy and descriptiveness of the identified models. Finally, we evaluate the approach on a set of selected examples.

In Plambeck et al. [23], we introduced the basic idea of the identification algorithm. In addition to that, we now have the following contributions 1) a discussion of SR in the context of hybrid system identification, 2) the identification of relevant parameters for the identification of hybrid systems using SR, and 3) a structured analysis of learning parameters considering multiple example systems including a physical simulation of a two-state power converter, a two tank system, and a static electrical circuit with multiple power sources.

The paper is structured as follows: in Section 2, we review related work on system identification, specifically for hybrid systems and SR. In Section 3, we introduce the necessary formal definitions. In Section 4, we revisit the algorithm presented in [23] and discuss the impact of parameters on the identification of hybrid systems. In Section 5, we perform an intensive parameter study. The algorithm and the parameter study are open source and available at [21]. Finally, in Section 6, we conclude the paper.

2 Related Work

Symbolic Regression (SR) is a method for regression and system identification that aims to find a symbolic expression that matches a given data set. Contrary to traditional regression methods, SR is not restricted to a specific set of functions or structure of an expression

like, e.g., polynomial regression, but uses a set of basic operators to construct complex expressions freely. The development of SR has been supported by the advancement of genetic programming, which is commonly used to implement SR algorithms [15]. In addition to genetic programming, there exist further approaches for SR, e.g., using deep reinforcement learning [20] or lattices [6].

Genetic programming methods like SR have been applied to a wide range of problems, including the identification of physical concepts from data [26], hybrid dynamical systems [18] mining the expression of diagnosis indicators [11].Thus, SR is one of several methods for system identification together with a broad range of other identification methods such as linear or nonlinear regression, neural networks, or kernel-based methods. Schoukens et al. [27] provides a comprehensive overview on these methods. In Koza [14], general hypotheses about the capabilities and convergence of SR are already discussed. Other and recent works investigate the influence of specific parameters such as the population size and number of generations on the performance of SR [16].

Hybrid systems, exemplified by hybrid automata, are the focus of this paper. Existing approaches for identification of hybrid systems from data, present several different strategies. Among them are clustering methods [3], machine learning with neural networks [19], and linear inequalities [29]. The general procedure, separating the process of learning in multiple steps as shown in Fig. 1, is similar in all of these approaches. Nevertheless, the detection of transitions in these methods is based on the similarity of signals, e.g., based on windows of the observations [29] or on distances between signals [3] or in the frequency domain [19]. Here, our approach using SR offers the opportunity to detect the decision points for transitions directly based on (an estimate of) the dynamics of the observed data.

3 Preliminaries

In this section, we introduce the basics of Symbolic Regression (SR) and formally define hybrid systems and system observations. We follow the presentation given in [23].

3.1 Symbolic Regression

SR is a machine learning technique that aims to find a mathematical expression describing the relationship between multiple input variables and a single output variable, i.e., SR searches a function r with $o = r(\mathbf{i})$, where $\mathbf{i}$ are the input variables and o is the output variable. The function r is represented as a mathematical expression in terms of elementary functions and operators [14]. The search for the best symbolic expression is guided by a fitness metric that evaluates the quality of candidate expressions on a data set. Learning algorithms typically represent the expression as a tree structure, where nodes represent basic operators and leaves represent variables or constants.

▶ **Definition 1** (SR Search & Solution Space). *The search space $\mathcal{E}$ is the set of possible tree structures defined by a set of operators as well as a set of variables and constants. The solution space $\mathcal{S} \subseteq \mathcal{E}$ is the set of expressions, i.e., tree structures, whose fitness is above a predefined threshold.*

The search space is often constrained by limiting the maximum depth of the tree, the number of nodes, or the set of operators [9, 20].

In the scope of this work, we use the framework PySR for SR, which uses Genetic Programming (GP) [14, 9]. Using GP, the search for the best symbolic expression is performed by evolving a population of candidate expressions over so-called *iterations* as

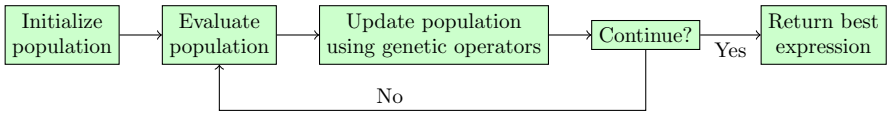


Figure 2 Genetic Programming Algorithm for Symbolic Regression.

shown in Fig. 2. The population is initialized with random expressions. In each iteration, the population is evaluated using a fitness function. New solutions are generated by applying genetic operators, such as mutation and crossover, to the best candidates. The process is repeated until a stopping criterion is reached, e.g., predefined number of iterations.

Nevertheless, for GP-based algorithms, there is no guarantee that the learned expression is the exact one, because the underlying search process is randomized, there exist mathematically or logically equivalent solutions, and learning data might be noisy. Thus, SR is most useful in domains where close approximations to an explicit solution exist and are useful [24].

SR has shown to rapidly approach the close neighborhood of the optimal solution, struggling only in converging to a precise result [14]. Thus, we can expect that the learned expressions represent the most dominant dynamics of the system and are compact. Several studies show that the performance of SR improves by using a suitable parametrization. The population size and the number of iterations are usually considered the most crucial parameters – usually a larger population and number of iterations leads to higher accuracy [10, 17, 16].

Another known problem in the evolution of learned expressions in SR is *bloat*. Bloat describes the growth of the learned expression without a significant improvement in the fitness. Bloat is often addressed by using a parsimony pressure [24]. In this case, SR solves a multi-objective optimization problem, where the fitness of the expression is not only determined by the quality of the approximation but also by the size of the expression. The *parsimony coefficient* regulates the trade-off between the two objectives.

3.2 Hybrid Systems

The literature identifies types of dynamical systems, which are hybrid, i.e., involve discrete and continuous dynamics. These types of systems illustrate different levels of model expressiveness needed to represent the dynamics.

- Jump linear systems are described by stochastic processes [8]. They are often represented by a Markov chain where modes are associated with different linear systems.
- Piecewise affine systems for which flow functions are affine functions [7] and the state space is partitioned into polyhedral regions.
- Switched systems for which the flow functions are general continuous functions.
- Mixed logical dynamical systems include logical variables to model discrete events or conditions in the system description [28].
- Projected dynamical systems extend the expressiveness of flow functions to non-linear expressions [13].

Here, building upon Branicky [5], we use a generalizing definition of hybrid systems which incorporates all the known system types listed above.

▶ **Definition 2** (Hybrid System [5]). *A Hybrid System is defined by a 6-tuple* $(X, Q, \mathcal{F}, \mathcal{T}, \Sigma, R)$ *where*

- $X = \{x_1, x_2, ..., x_n\} = I \cup O \cup S$ *is the set of system variables which consist of input variables* I*, output* O *and state variables* S*. Derivatives of variables may form individual variables in* X*.*
- Q *is the set of modes.*
- $\mathcal{F}$ *is the set of flow functions. Every flow function* $f_q \in \mathcal{F}$ *defines the change of the state variables* S *as well as the current output variables* O *over the continuous time* t *within the mode* q *based on the current values of state variables and the external inputs* I*, i.e.,* $[O(t), \dot{S}(t)] = f_q(S(t), I(t), t)$*.*
- $\mathcal{T} : Q \times \Sigma \to Q$ *defines transitions between modes* Q*. A transition is triggered if the corresponding event* $\sigma \in \Sigma$ *is active.*
- Σ *is a set of events leading to transitions between modes. Each event is guarded by a set of conditions on the variables in* X*. A transition is triggered if the conditions are met.*
- R *is a reset relation* $R : Q \times \Sigma \times X \to X$ *capturing discontinuous changes of the internal variables.*

A hybrid system, according to this definition, combines discrete and continuous behavior. Discrete behavior is captured by the discrete modes Q and transitions $\mathcal{T}$, while the flow functions $\mathcal{F}$ describe the continuous dynamics. Transitions are usually triggered by conditions on the variables or discrete control signals for mode transitions. In the scope of this paper, the goal is to identify a model of a real system according to Definition 2. We focus on the identification of modes and flow functions while excluding the construction of conditions. This usually implies that transition conditions are defined by external control signals. Within this scope, *dynamics* define the physical behavior of the real system. In the abstraction given by the model, i.e., the hybrid system, the dynamics are represented by the flow functions of the *modes*. Finally, observations of dynamics, i.e., changes in the values of the variables, are considered as *trajectories* defined as an observation in form of a multi-dimensional time series of the variables $X = \{x_0, x_1, \ldots, x_n\}$.

4 Identification of Hybrid Modes

In this section, we first revisit the approach presented in Plambeck et al. [23] and identify the parameters which are used in the parameter study.

4.1 Overview & System Properties

Our approach might model all of the system types encompassed by Definition 2, while we focus here on Jump Linear Systems, Piecewise Affine Systems, and Switched Systems, i.e., systems with continuous flow functions. The most characteristic property of hybrid systems is the combination of *continuous* dynamics (defined by the flow functions $\mathcal{F}$) with *discrete* modes (Q). To learn both parts, identification of hybrid systems includes multiple subproblems as shown in Figure 1 and as similarly introduced in Saberi et al. [25]:

1. detection of discrete transitions between dynamics, separating the trajectories into segments,
2. grouping of segments with identical dynamics, forming discrete modes,
3. identification of the continuous dynamics for each mode, i.e., the flow functions of $\mathcal{F}$,
4. model construction, i.e., accumulation of the results in a single hybrid system.

The modeling strategy covers steps **1.** to **3.** in which steps **2.** and **3.** are combined, both based on SR. Our approach, hence, involves two algorithms: one algorithm to detect the transitions between modes (*segmentation*) and, further on, a second algorithm to group and identify the flow functions of modes using the segmented trajectories (*grouping*). We assume a positive residence time in each of the hybrid modes. This residence time should provide sufficiently many data samples such that the original dynamics can be reconstructed.

4.2 Identification Algorithm

For the segmentation step, we begin with a small window of observed data, learning a symbolic expression. The window is gradually enlarged until the expression's fitness declines, indicating a *decision point* where a mode transition occurs. With each window increase, the expression adapts incrementally to capture changes in the dynamics. Pseudocode for segmentation is shown in Algorithm 1.

Algorithm 1 Detection of Mode Transitions: given an observed trajectory of the system, we process over this trajectory using a window (Line 4). Initially, this window covers a fixed initial length at the beginning of the trajectory. As long as the segmentation criterion is fulfilled, the window is extended to the right (Line 6). When the segmentation criterion is no longer met, a mode transition is detected, and the window is stored as a segment in the set $\mathcal{T}$. In the inner loop, the symbolic expression is learned incrementally, i.e., n_{update}-many iterations of the SR are performed [23].

Data: trajectory
Result: $\mathcal{T}$, *expressions*
1 $i_{start} \leftarrow 0$; $i_{end} \leftarrow l_{init}$; $n \leftarrow n_{init}$;
2 **while** $i_{end} < len(trajectory)$ **do**
3 **while** *segmentationCriterion fulfilled* **do**
4 window $\leftarrow$ trajectory$[i_{start}, \min(i_{end}, len(\text{trajectory}))]$;
5 learnExpression(window,n);
6 $i_{end} \leftarrow i_{end} + l_{step}$;
7 $n \leftarrow n_{update}$;
8 **end**
9 $\mathcal{T} \leftarrow \mathcal{T} \cup \{\text{window}[0, \text{end} - l_{step}]\}$;
10 $i_{start} \leftarrow i_{end} - l_{step}$; $i_{end} \leftarrow i_{start} + l_{init}$; $n \leftarrow n_{init}$;
11 resetSR;
12 **end**

In the subsequent grouping step, SR is reused to learn expressions on unions of the previously detected segments. When the loss of combined segments decreases compared to individual segments, they are grouped, identifying the mode. By this segments with the same dynamics, describable by the same flow functions, are grouped. Pseudocode for grouping is in Algorithm 2.

From this review of the algorithms, we find the following set of parameters as given in Table 1. The segmentation and grouping criteria as given in Plambeck et al. [23] are used.

5 Experiments & Parameter Study

In this section, we evaluate the feasibility and capability of SR for hybrid system identification using our proposed algorithm. Thus, the evaluation scenario focuses the learning step. We use a single trace observed on an example system, for which we know both, the ground truth

Algorithm 2 Grouping of Modes: the input to the grouping is the set of detected segments $\mathcal{T}$. The first segment is a first candidate group as stated in Line 1. Afterward, we iterate for every segment in $\mathcal{T}$ in Line 2 over all known groups in Line 4. The current segment and the current group are combined to one data set and an expression is learned (see Line 5). If the loss of the learned expression is small, the current segment is included in the current group (see Lines 6 and 7). If no matching group is found for the current segment, the segment forms a new group as stated in Line 11 [23].

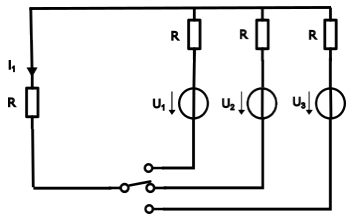
```
   Data: 𝒯
   Result: G, expressions
 1 G ← {S[0]};
 2 for s ∈ 𝒯 do
 3 |   groupFound ← False;
 4 |   for g ∈ G do
 5 |   |   exp, fit ← learnExpression(s ∪ g, n);
 6 |   |   if groupingCriterion fulfilled then
 7 |   |   |   g ← s ∪ g;
 8 |   |   |   expressions[g] ← exp; groupFound ← True; break;
 9 |   |   end
10 |   end
11 |   if not groupFound then
12 |   |   G.append(s);
13 |   end
14 end
```

Table 1 Parameters of the Learning Process.

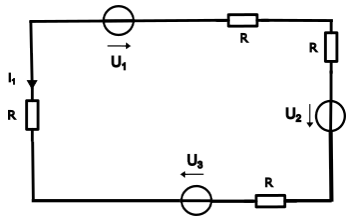
Symbol	Occurrence	Description
l_{init}	Segmentation	initial window size when learning an expression
l_{step}	Segmentation	step-width for extending the window
n_{init}	Segmentation	number of iterations of SR when learning an expression
n_{update}	Segmentation	number of iterations of SR for updating the expression on an extension
τ	Segmentation	threshold for the segmentation criterion
φ	Grouping	relaxation parameter for the grouping criterion
n_g	Grouping	number of iterations of SR when learning on grouped data
ρ_s, ρ_g	General SR	Parsimony coefficient (length-accuracy trade-off) for segmentation and grouping
p_s, p_g	General SR	Population size for segmentation and grouping

decision points of the trace and the ground truth expressions of the flow functions of the systems. Our analysis focuses the accuracy, that the SR-based learning is able to achieve compared on the learning trace and with respect to the ground truth information. The evaluation of the learned model on an evaluation data set is out of scope here.

In the following, we first showcase the usability on two simple examples which we will use for a further discussion on system and model properties later on. Afterwards, we present an intensive parameter study for the presented algorithm on two real-world examples. This study provides additional insights in the usability of SR for the identification of hybrid systems and aligns with the previous introduction of SR. Furthermore, the study provides indications on how to choose parameters for to be learned systems.

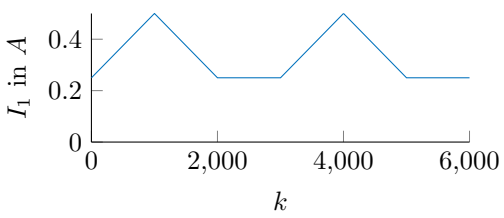


(a) Version 1 – Switched sources.

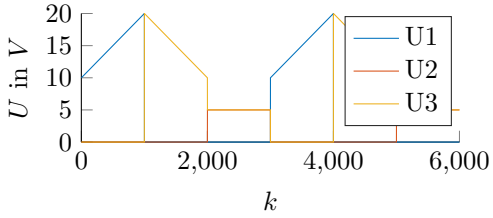


(b) Version 2 – Connected sources.

Figure 3 Passive Electrical Circuits.



(a) Output current I_1.



(b) Input voltages U_1, U_2, U_3.

Figure 4 Current and Voltage of Circuit 2.

5.1 Usability of SR-Based Hybrid System Identification on Simple Examples

We consider two versions of a passive electrical circuit, which are shown in Figure 3. In both examples, the goal is to learn the flow functions $o(t) = f_q(\mathbf{i}(\mathbf{t}))$, where the inputs $\mathbf{i} = [U_1, U_2, U_3]$ are the voltage levels of the sources and the output $o = I_1$ is the main current in the circuit. The first circuit in Fig. 3a is described by the equation. Depending on the position of the switch, one of the three sources is selected. This is a simple example of a hybrid system with three modes.

$$I_1 = \begin{cases} \frac{U_1}{2 \cdot R}, & \text{switch in 1st position} \\ \frac{U_2}{2 \cdot R}, & \text{switch in 2nd position} \\ \frac{U_3}{2 \cdot R}, & \text{switch in 3rd position} \end{cases} . \tag{1}$$

The second circuit shown in Fig. 3b also contains three sources, but here all of them are connected within the circuit. Fig. 4 shows the output current I_1 and the three voltages over the discrete sampling points k. From visual inspection of I_1, we might assume three operational modes of the system. Nevertheless, the different appearances of I_1 solely result from changes in the input signals. In fact, the system is completely described by the equation

$$I_1 = \frac{U_1 + U_2 + U_3}{4 \cdot R}. \tag{2}$$

Even though this circuit does not show a hybrid behavior, we consider this as an interesting example, as one might assume different modes from visual inspection. Also, this example shows that whether multiple modes are needed or not can be ambiguous as both of the circuits might lead to identical observations. There are multiple approaches that could resolve this issue. One possibility is to choose the most compact representation.

For SR, both circuits use the same set of basic operators which contains addition, subtraction, multiplication, and division operators. The three voltages U_1, U_2, U_3 and the values of the resistance R are given as variables for learning an expression for I_1. Both

Table 2 Identification Results for the Simple Examples, True Positives (TP) and False Positives (FP) are given in percent, $|S|$ is the number of detected transitions, $|G|$ is the number of detected groups.

System	$\|S\|$	TP	FP	$\|G\|$	Learned Expressions
Circuit 1	5	100	0	3	Group 1 – $U_1/(2 \cdot R)$ Group 2 – $U_2/(2 \cdot R)$ Group 3 – $U_3/(2 \cdot R)$
Circuit 2	0	-	-	1	Group 1 – $(U_1 + U_2 + U_3)/(4 \cdot R)$

circuits use $R = 10\Omega$ for all resistors. The parameters for learning as listed in Table 1 are set to $l_{init} = 200$, $l_{step} = 100$, $n_{init} = 20$, $n_{update} = 5$, $l_{hist} = 1$, $\tau = 1 \cdot 10^{-7}$, $n_{group} = 20$, and $\varphi = 1.5$.

Table 2 shows the results of the identification process for the two circuits. For the first circuit all five transitions are detected correctly and no false positive, i.e., additional transitions are detected. For the second circuit, no transitions are detected, as the system is actually not hybrid. This shows that the approach is able to identify identical dynamics even though the visual inspection of the trajectories may suggest different modes as discussed for Figure 4.

The learned expressions for both circuits are equivalent to the ground truth expressions. Thus, leading to a mean-square error loss of zero for the predicted trajectories. The results show that the approach is able to identify the structure of the hybrid systems from data perfectly for simple examples.

5.2 Parameter Study

Having shown the usability of the SR-based algorithm on simple examples, we now present a parameter study on two real-world examples introduced in the following. The first example is the two tank system, which is a known benchmark system for hybrid systems. The second example is a power converter, which is a real-world system with a complex behavior.

5.2.1 Two Tank System

The two tank system [4] is a benchmark system for hybrid systems, which consists of two tanks with a pump pumping water into the first tank. A valve V_b regulates whether water flows from the first to the second tank. The system is described by the following differential equation:

$$\dot{h}_1 = \begin{cases} \frac{Q_p - C_{vb} \cdot \sqrt{h_1 - h_2}}{A}, & \text{if } h_1 > h_2, V_b \text{ open} \\ \frac{Q_p + C_{vb} \cdot \sqrt{h_2 - h_1}}{A}, & \text{if } h_1 \leq h_2, V_b \text{ open} \\ \frac{Q_p}{A}, & \text{if } V_b \text{ closed,} \end{cases} \tag{3}$$

where h_1 and h_2 are the heights of the water in the first and second tank, respectively, $\dot{h}_1$ is the derivative, i.e., the change in the water level, of the first tank. Q_p is the flow rate of the pump, C_{vb} is the valve conductance, and A is the cross-sectional area of the first tank. The simulation involves noise and a realistic controller which applies a zero-order hold to the observed height of the tank. The noise is additive white noise with a maximum amplitude of 10^{-6} for all sensors.

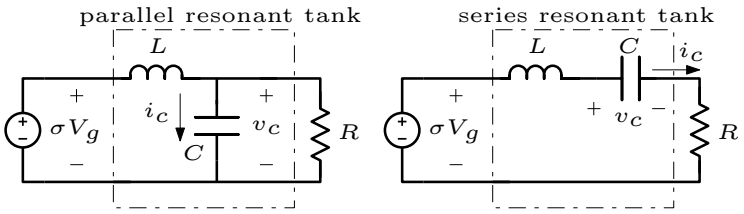


Figure 5 Series and Parallel Representation of the Power Converter [30].

In the modeling process, we consider two different versions of the two tank:

- Version 1 (with substitution): the pre-calculated term $\sqrt{|h_1 - h_2|}$, is given as an additional variable. The operators for learning involve addition, subtraction, multiplication, and division.
- Version 2 (without substitution): the pre-calculated term is not given as a variable. The operators for learning involve addition, subtraction, multiplication, division, and, additionally the square-root.

For both versions, variables for SR are the inflow Q_p and the height of the two tanks h_1 and h_2 as well as the constants C_b and A. The goal is to learn the derivative $\dot{h}_1$, i.e., the flow function which defines the state change $\dot{s}(t) = f_q(s(t), \mathbf{i}(t))$, with $s = h_1$ and $\mathbf{i} = [h_2, Q_p, C_b, A]$.

5.2.2 Power Converter

The power converter [30] is a real-world system that has a controlled input voltage $v_s = \sigma V_g$ where V_g is a constant input and σ is a switching variable which can be either 1 or -1. In addition to the voltage source, the circuit consists of a capacitor C, an inductance L, and a resistor R. The circuit can be either a parallel or a series configuration, as shown in Figure 5. As presented in [30], we use a transformed coordinate system to describe both configurations with the same equations. The system is described by the following differential equation:

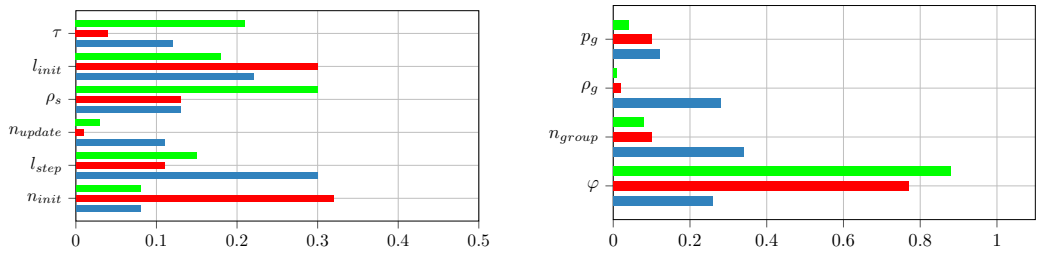
$$\dot{w} = \begin{pmatrix} 0 & \alpha \\ -\alpha & -\beta \end{pmatrix} w + \begin{pmatrix} 0 \\ \alpha \end{pmatrix} \sigma, \tag{4}$$

where $\alpha = \frac{1}{\sqrt{LC}}$ and $\beta = \frac{1}{RC}$ (parallel case) or $\beta = \frac{R}{L}$ (serial case). The transformed coordinates $w = [w_1, w_2]^T$ are constructed from the quantities in Figure 5 as

$$w_1 = \frac{v_c}{V_g} \quad \text{and} \quad w_2 = \frac{1}{V_g}\sqrt{\frac{L}{C}} \cdot i_c.$$

Our goal is to model the state variable w_2. The inductance, capacity, and resistor have the values $R = 400\Omega, L = 8\mu H, C = 10.5nF$ and $V_g = 20V$.

For SR, the power converter uses addition, subtraction, multiplication, and division as well as the square-root as basic operators. Variables for learning are w_1, w_2, and the continuous time t of a sampling point. Constants are not given as variables, but are estimated by the learner. The goal is to learn an expression for $\dot{w}_2$, where the derivation is numerically performed. Thus, we learn the flow function which defines the state change $\dot{s}(t) = f_q(s(t), \mathbf{i}(t), t)$, with $s = w_2$ and $\mathbf{i} = [w_1]$.



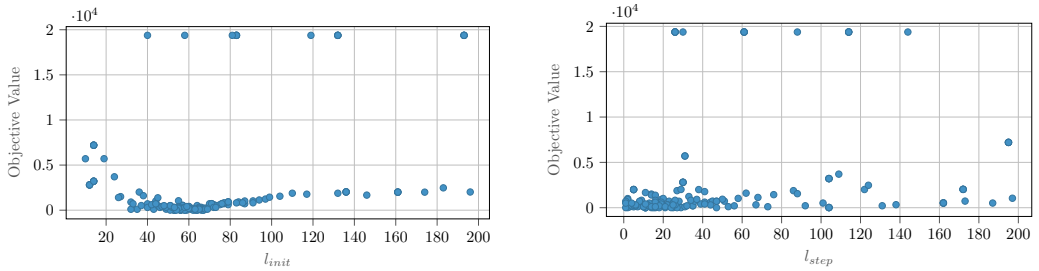
(a) Segmentation. **(b)** Grouping.

Figure 6 Parameter importances, Two Tank System with Substitution (green), Two Tank System without Substitution (red), Power Converter (blue).

5.2.3 Experimental Results

For both systems, we analyze the parameters l_{init}, l_{step}, n_{init}, n_{update}, and τ for the segmentation step and n_{group}, φ, and p_g for the grouping step. Additionally, the parsimony coefficients ρ_s and ρ_g for SR during the segmentation and grouping step, respectively, are analyzed. The parameter study is executed with the hyperparameter optimization library Optuna [1] with at least 100 trials for every study. The optimization uses an *objective function*. Parameter importances are found with the fANOVA approach [12] of Optuna. More information on optuna, the objective functions and fANOVA can be found in the project's repository [21].

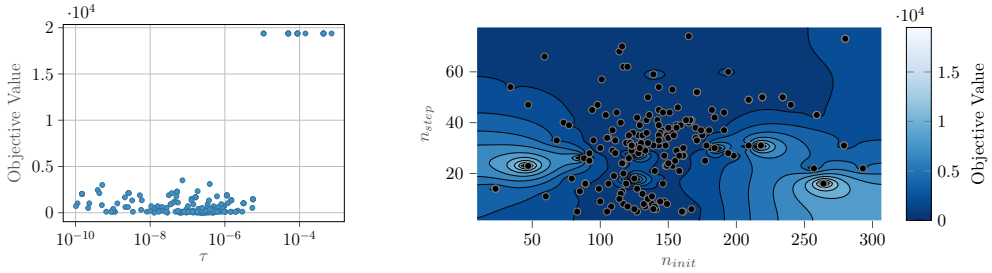
The parameter importances, as calculated with Optuna using the fANOVA approach [12], show that for most examples, the initial window width l_{init} as well as the length for extending the window l_{step} are relevant. Figure 7 shows these two parameters against the objective value. Note, that this and the following plots always show all runs, i.e., also other than the presented parameters are varied over the runs. We observe, that l_{init} against the objective value has a sweet spot around 60, which is about the mean number of samples, that the system stays within a mode. These observations are intuitive as the best initial window size would be the one that captures exactly one occurrence of a mode. This implies that prior knowledge or a good assumption on the expected time spent in a mode improves the model learning procedure. For l_{step}, the plot indicates that smaller values usually lead to smaller objective values. Thus, a small step size when increasing the window identifies the decision points more accurately. Still, a smaller step width leads to longer runtime.



(a) Start width l_{init} against objective value. **(b)** Step width l_{step} against objective value.

Figure 7 Segmentation: parameters against objective value for Two Tank without substitution.

The number of generations n_{init} and n_{step} are also relevant parameters. Figure ref-fig:twotanksubsegiter shows the two parameters in a contour plot of the objective value, where dark colors indicate good, i.e., small objective values. For this example, the number of initial iterations n_{init} is best around 100 to 150. This number of iterations allows to pre-learn



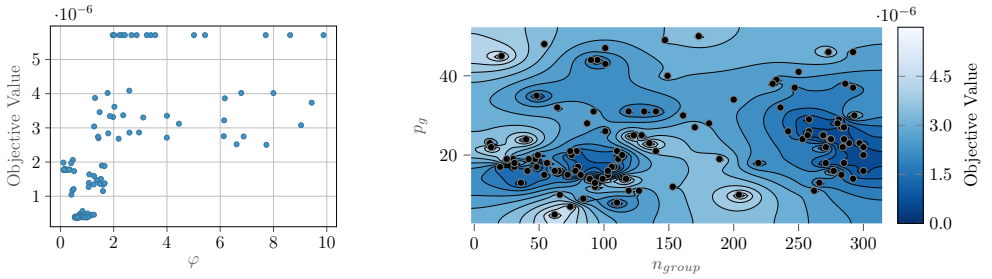
(a) Saturation τ against objective value. **(b)** Contour of objective value against n_{init}, n_{step}.

Figure 8 Segmentation: parameters against objective value for Two Tank with substitution.

the dynamics without overfitting such that the inclusion of new data is possible on window extension. The number of generations for updating the population n_{step} has low influence on the objective value. To reduce runtime, we could, thus, choose a small value for n_{step}.

The saturation threshold for the segmentation criterion τ shows a clear impact as shown in Figure 8a. As introduced earlier, this threshold assures that the extension of the window is continued as long as the loss stays below the threshold. Thus, the value of τ has to be below a certain level to correctly identify decision points.

For the grouping step, the grouping factor φ is the most important parameter, because the choice of this parameter clearly separates experiments with a low and a high objective value. We also have a clear dependency of the objective in other parameters such as the number of iterations n_{group}. Figure 9b shows the number of iterations and the size of the population against the objective value. A larger number of iterations is preferable, because a higher number of generations allows for a better exploration of the solution space. The size of the population has less influence on the objective value. Figure 9a shows the factor φ for the grouping criterion against the objective value. Here, we find a clear minimum around $\varphi = 1$, i.e., where we require a strict decrease in the loss when adding a segment to a group.



(a) Factor φ against objective value. **(b)** Contour plot of objective value against n_{group},p_g.

Figure 9 Grouping: parameters against objective value for Converter.

5.3 Accuracy of Predicted Trajectories

In the last step, we showcase the accuracy of the learned models with good parametrization with respect to our objective functions. The results are shown in Table 3.

We observe that the learned expressions are not identical to the ground truth expressions, but represent the most dominant behavior. This aligns with the known property of SR which tends to converge to solutions close to the optimum, but matching the exact correct expression is difficult especially if the solution space is large, i.e., the expression to be learned is complex. A deeper discussion and comparison of the predicted and the original trajectories can be found in [23].

Table 3 Identification Results, True Positives (TP) and False Positives (FP) are given in percent, $|S|$ is the number of detected transitions, $|G|$ is the number of detected groups.

System	$\|S\|$	TP	FP	$\|G\|$	Learned Expressions		Loss
Converter	4	100	0	2	Group 1	$3.37 \cdot 10^{-3} - 4.71 \cdot 10^{-3} \cdot w1$	$4.76 \cdot 10^{-7}$
					Group 2	$-6.28 \cdot 10^{-3} \cdot \sqrt{w1 + 0.396}$	
Two Tank 1	27	92.9	3.7	3	Group 1	Q_p/A	$5.26 \cdot 10^{-6}$
					Group 2	$(-Cv_{vb} \cdot \sqrt{\lvert h_1 - h_2 \rvert} + Q_p)/A$	
					Group 3	$(-C_{vb} + Q_p/h_1)/A$	
Two Tank 2	28	92.9	0	3	Group 1	$Q_p/A - \sqrt{Q_p} \cdot h_1$	$4.68 \cdot 10^{-6}$
					Group 2	Q_p/A	
					Group 3	$\sqrt{C_{vb}} - A$	

6 Conclusion

In this paper, we provide a deep discussion of symbolic regression for hybrid system identification. Revisiting a proposed method for hybrid system identification with symbolic regression, separated in two algorithms, we cover three major aspects. First, we discuss known properties of symbolic regression regarding accuracy and convergence and put them in the context of hybrid system identification. Furthermore, we provide an intensive parameter study of the two identification steps. We see that a higher number of generations leads to more accurate models. Furthermore, prior knowledge on the system behavior can support the learning process. The last part of the paper is dedicated to a discussion of system types in the regime of hybrid systems and within the context of symbolic regression. We argue that the complexity in the expression of dynamics and the number of modes of a model form a solution space where large models with simple expressions form similarly accurate models as small models with complex expressions. The choice of model size and expression complexity, thus, can be seen as a design decision during model learning.

References

1 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. arXiv:1907.10902.

2 Rajeev Alur. Principles of Cyber-Physical Systems. Technical report, MIT Press, 2015.

3 Nathalie Barbosa Roa, Louise Travé-Massuyès, and Victor H. Grisales-Palacio. Dyclee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94:162–186, 2019. doi:10.1016/J.PATCOG.2019.05.024.

4 B. Ould Bouamama, R. Mrani Alaoui, P. Taillibert, and M. Staroswiecki. Diagnosis of a two-tank system. Technical report, Intern Report of CHEM-project, USTL, 2001.

5 Michael S. Branicky. *Introduction to Hybrid Systems*, pages 91–116. Birkhäuser, Boston, 2005.

6 Kevin René Broløs, Meera Vieira Machado, Chris Cave, Jaan Kasak, Valdemar Stentoft-Hansen, Victor Galindo Batanero, Tom Jelen, and Casper Wilstrup. An approach to symbolic regression using feyn. *arXiv*, 2021. arXiv:2104.05417.

7 Frank J. Christophersen. *Piecewise Affine Systems*, pages 39–42. Springer, Berlin Heidelberg, 2007.

8 Oswaldo Luiz Valle Costa, Ricardo Paulino Marques, and Marcelo Dutra Fragoso. *Markov Jump Linear Systems*, pages 1–14. Springer, London, 2005.

9 Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023. arXiv:2305.01582, doi:10.48550/arXiv.2305.01582.

10 Robert Feldt and Peter Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In *Genetic Programming*, 2000.

11 Louis Goupil, Elodie Chanthery, Louise Travé-Massuyès, and Sébastien Delautier. Tree based diagnosis enhanced with meta knowledge. In *International Workshop on Principles of Diagnosis (DX)*, 2023.

12 F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International Conference on Machine Learning (ICML)*, 2014.

13 Hassan K. Khalil. *Nonlinear systems*. Pearson Education International, 3. edition, 2000.

14 John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1994.

15 Gabriel Kronberger, Lukas Kammerer, and Michael Kommenda. Identification of dynamical systems using symbolic regression. *arXiv*, 2021. `arXiv:2107.06131`.

16 William B. Langdon. Genetic programming convergence. In *Genetic and Evolutionary Computation Conference Companion*, 2022.

17 Thomas Loveard and Vic Ciesielski. Genetic programming for classification: An analysis of convergence behaviour. *Lecture Notes in Artificial Intelligence*, 2557:309–320, 2002. `doi:10.1007/3-540-36187-1_27`.

18 Daniel L. Ly and Hod Lipson. Learning symbolic representations of hybrid dynamical systems. *Journal of Machine Learning Research*, 13(115):3585–3618, 2012. `doi:10.5555/2503308.2503356`.

19 Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *AAAI Conference on Artificial Intelligence*, 2012.

20 Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

21 Swantje Plambeck. Symbolic Regression for Hybrid Automata, 2024. Software, swhId: `swh:1:dir:ee3af8e7fcea2b4be54dffa349ab4c87cdb15759` (visited on 2024-11-12). URL: `https://github.com/TUHH-IES/SymbolicRegression4HA`.

22 Swantje Plambeck, Aaron Bracht, Nemanja Hranisavljevic, and Goerschwin Fey. Famos–fast model learning for hybrid cyber-physical systems using decision trees. In *International Conference on Hybrid Systems: Computation and Control*, 2024.

23 Swantje Plambeck, Maximilian Schmidt, Audine Subias, Louise Travé-Massuyès, and Goerschwin Fey. Dynamics-based identification of hybrid systems using symbolic regression. In *Software Engineering and Advanced Applications (SEAA)*, 2024.

24 Riccardo Poli, William B Langdon, and Nicholas F McPhee. *A Field Guide to Genetic Programming*, volume 10. Springer, 2008.

25 Iman Saberi, Fathiyeh Faghih, and Farzad Sobhi Bavil. A passive online technique for learning hybrid automata from input/output traces. *ACM Transactions on Embedded Computing Systems*, 22(1), October 2022. `doi:10.1145/3556543`.

26 Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

27 Johan Schoukens and Lennart Ljung. Nonlinear system identification: A user-oriented road map. *IEEE Control Systems Magazine*, 39(6):28–99, 2019.

28 E. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on Automatic Control*, 26(2):346–358, 1981.

29 Xiaodong Yang, Omar Ali Beg, Matthew Kenigsberg, and Taylor T. Johnson. A framework for identification and validation of affine hybrid automata from input-output traces. *ACM Transactions on Cyber-Physical Systems*, 6(2):1–24, 2022. `doi:10.1145/3470455`.

30 Nicola Zaupa, Luis Martínez-Salamero, Carlos Olalla, and Luca Zaccarian. Hybrid control of self-oscillating resonant converters. *IEEE Transactions on Control Systems Technology*, 31(2):881–888, 2023. `doi:10.1109/TCST.2022.3179948`.

Using Multi-Modal LLMs to Create Models for Fault Diagnosis

Silke Merkelbach
Fraunhofer IEM, Paderborn, Germany

Alexander Diedrich
Helmut-Schmidt-University, Hamburg, Germany

Anna Sztyber-Betley
Warsaw University of Technology, Poland

Louise Travé-Massuyès
LAAS-CNRS, University of Toulouse, France

Elodie Chanthery
LAAS-CNRS, INSA, University of Toulouse, France

Oliver Niggemann
Helmut-Schmidt-University, Hamburg, Germany

Roman Dumitrescu
Advanced Systems Engineering, Paderborn University, Germany

Abstract

Creating models that are usable for fault diagnosis is hard. This is especially true for cyber-physical systems that are subject to architectural changes and may need to be adapted to different product variants intermittently. We therefore can no longer rely on expert-defined and static models for many systems. Instead, models need to be created more cheaply and need to adapt to different circumstances. In this article we present a novel approach to create physical models for process industry systems using multi-modal large language models (i.e ChatGPT). We present a five-step prompting approach that uses a piping and instrumentation diagram (P&ID) and natural language prompts as its input. We show that we are able to generate physical models of three systems of a well-known benchmark. We further show that we are able to diagnose faults for all of these systems by using the Fault Diagnosis Toolbox. We found that while multi-modal large language models (MLLMs) are a promising method for automated model creation, they have significant drawbacks.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Fault Diagnosis, Large Language Models, LLMs, Physical Modelling, Process Industry, P&IDs

Digital Object Identifier 10.4230/OASIcs.DX.2024.31

Category Short Paper

Supplementary Material *Collection*: `https://github.com/silkeme/DX24_Model_Creation_LLMs`

Funding *Silke Merkelbach*: This work is supported by the German Federal Ministry for Economic Affairs and Climate Action under grant 03EN4004B.
Anna Sztyber-Betley: This project was partially funded by a research grant from the Scientific Council of the Discipline of Automation, Electronics, Electrical Engineering and Space Technologies of Warsaw University of Technology granted in 2023.
Louise Travé-Massuyès: The work is supported by ANITI through the French "Investing for the Future – P3IA" program under the Grant agreement n°ANR-19-P3IA-0004.
Elodie Chanthery: The work is supported by ANITI through the French "Investing for the Future – P3IA" program under the Grant agreement n°ANR-19-P3IA-0004.

Acknowledgements This work has benefited from participation in Dagstuhl Seminar 24031 "Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis".



© Silke Merkelbach, Alexander Diedrich, Anna Sztyber-Betley, Louise Travé-Massuyès, Elodie Chanthery, Oliver Niggemann, and Roman Dumitrescu;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Model-based fault diagnosis requires the availability of logical or physical models to reason from observations that deviate from normal behaviour to possible root causes (i.e. faults) [26]. The major drawback common to all model-based fault diagnosis approaches is the limited availability of complete models [5] that exactly describe the behaviour of the underlying system. Obtaining a model is usually an expensive process, requires the close cooperation of experts, and results in most cases in static formulations with respect to the system's architecture. However, changes in architecture and changes in system parameters are common in today's cyber-physical systems (CPS), which places strong requirements on models and thus makes the application of fault diagnosis methods quite rare.

The challenge in model creation is to reliably detect the physical dependencies between components along with each component's behaviour. Formally, this is known as the system identification problem. According to Aguirre [1], system identification is done using a set of measured data to find a mathematical model $\mathcal{M}$ that represents a system in some meaningful way. To build $\mathcal{M}$ exclusively from data is called a black-box identification problem. In grey-box problems, besides data, there is some other source of external information about the system. In this article, we use a white-box approach to system identification, using detailed knowledge about the system's physical laws and underlying mechanisms.

Several authors have attempted to solve the system identification problem for fault diagnosis through black-box and grey-box approaches [7, 18]. Frisk et al. [10] have come up with a semi-automatic white-box approach to create fault diagnosis models from physical equations. But they rely on exact specifications of the system's components, connections, and their governing equations, which we are trying to avoid in this article. Plambeck et al. [23] propose to learn models of hybrid systems using symbolic regression. Their method leverages the power of genetic programming to automatically discover interpretable mathematical models in the form of hybrid systems from observed data. But they require a historical dataset, which is something we want to avoid. We also do not want to rely on complex and hard-to maintain ontologies that some authors have used for this task [11]. So far, large-language models (LLMs) have not been broadly adopted by the fault diagnosis community yet. Some [13, 28] have presented the usage of LLMs for software fault localization. Others have shown how to generate equations with LLMs [8, 24]. Balhorn et al. [2] have used LLMs to correct P&ID diagrams, but have not attempted to generate any kinds of models from the diagrams. Hirtreiter et al. [12] attempted automatic generation of control structures using LLMs. Ogundare et al. [19] have analysed the resilience of LLMs to create system models. But their models are limited to single equations that have no automatic dependencies between each other. Kato et al. [14] extract equations from a large number of scientific documents and create physical models out of them by judging their equivalence using a pre-trained LLM and defining requirements the physical model has to meet. But they neither use P&IDs nor do they create models for fault diagnosis. Peifeng et al. [21] presented the application of fault diagnosis with LLMs in aviation assembly, but this approach requires additional knowledge graphs. The problem of automatic knowledge extraction from P&IDs has been considered in recent works using deep learning and graph search [16]. Sinha et al. [25] propose a solution to detect tables and extract descriptions therein from P&ID diagrams automatically.

In this article we present a novel approach to use common design documentation for cyber-physical systems in the process industry in the form of piping and instrumentation diagrams (P&IDs) to generate a physical model using a multi-modal large language model (MLLM). P&IDs picture the components, sensors, the structure of the system, and the names

of the elements, providing a lot of helpful information for fault diagnosis. MLLMs are, among other things, able to handle images and text as input and create text as output. As MLLM we use OpenAI's GPT4 (ChatGPT) [20] with prompt engineering. As a direct continuation of some of our earlier work [17], we now present a more extended approach, do not provide the equations to the model, and, given the physical model created by ChatGPT, use the Fault Diagnosis Toolbox (FDT) [10] to generate analytical redundancy relations (ARRs). The ARRs are used to: i) compute the fault signature matrix [9]. This is a validation technique to establish how well we can diagnose potential faults. ii) compute residuals to determine faulty signals and thus discriminate faults.

With our contribution we want to present a novel diagnosis methodology for the fault diagnosis (DX) and fault detection and isolation (FDI) communities which uses LLMs, in particular MLLMs, to automatically generate physical models from existing design documentation. Our methodology consists of five steps, takes existing P&IDs as input, and creates models, that can be directly used as input for the fault diagnosis toolbox which can then be applied to create and evaluate ARRs for fault diagnosis. As expected, MLLMs still have severe limitations and our approach lacks generalisation to systems that are outside of well-known water tank benchmarks and similar process industrial examples. However, using our method in a generate-and-test approach we believe that practitioners can generate a statistically significant number of models and then select those that i) compile and ii) have a sufficiently high isolability. We believe that our approach has strong benefits in practical use-cases where models of systems with known components need to be created cheaply. Our method enables practitioners to generate new models whenever an updated system documentation exists. It also directly enables the generation of residuals from the identified model and thus decreases the costs to create self-diagnosing and resilient systems.

We have evaluated our approach on a one-tank, one three-tank, and one four-tank system of the benchmark provided by Balzereit et al. [3]. We first evaluated each system qualitatively and then used it together with the FDT to perform a quantitative evaluation.

2 Background

In this section, we explain the basis of our physical models, in particular how they integrate faulty behaviour, how we process observations from the system, and how we create diagnosis tests.

▶ **Definition 1** (Measurement, Control Value). *A measurement is a single measured value from a CPS sensor $z_{s_i}(t) \in \mathbb{R}$. A control value is a single value from a CPS actuator $z_{c_i}(t) \in \mathbb{R}$.*

Measurements and control values form the sets of known variables $\mathcal{Z} = \mathcal{Z}_s \cup \mathcal{Z}_c$. For simplicity, time notation is often omitted, assuming values are taken concurrently. We thus write z_i instead of $z_i(t)$. There are two types of faults: additive and multiplicative.

▶ **Definition 2** (Additive Fault). *An additive fault is a fault that changes the value of some measurement or control value $z_i \in \mathbb{R}$ by introducing an offset $f_i \in \mathbb{R}$ in an additive way, so that the sensor/actuator reads $z_i' = z_i + f_i$ instead of z_i.*

▶ **Definition 3** (Multiplicative Fault). *A multiplicative fault is a fault that changes the value of some parameter $p_i \in \mathbb{R}$ by introducing a factor $f_i \in \mathbb{R}$ that impacts on the system's dynamics.*

Additive faults occur when some bias is introduced into the system, such that the measurement is changed by some fixed amount. For example, the wrong configuration of some analog-to-digital converter, or the introduction of the wrong control voltage may lead

to additive faults. Conversely, multiplicative faults, also called parameter faults, affect the system's dynamics and hence, the stability of the system. They may lead to components limited in their functionality. For example a valve that does not open anymore, may limit its throughput to only 30%.

A physical model describes the information flow within a CPS. Through the use of physical relations, measurement values and intermediate values can be predicted.

A system model $\mathcal{M}(z, x, f)$ generally involves non observable or unknown variables, also named state variables, denoted x, and known variables z as defined previously. State variables x decompose in differential state variables x_1 and algebraic state variables x_2. The faults may be represented explicitly through a specific parameter vector f. The sets of known variables, unknown variables, and faults are denoted by $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{F}$ respectively. Note that in this paper, we limit ourselves to continuous dynamics physical systems. A typical model, known as a *state-space model*, may be formulated in the temporal domain as follows :

$$\mathcal{M}(z, x, f) : \begin{cases} dx_1(t)/dt = h(x_1(t), x_2(t), z_c(t), f), \text{ with } x_1(t_0) = x_0 \\ 0 = l(x_1(t), x_2(t), z_c(t), f) \\ z_s(t) = g(x_1(t), x_2(t), f). \end{cases} \tag{1}$$

where $x_1(t) \in \mathbb{R}^{n_1}$ and $x_2(t) \in \mathbb{R}^{n_2}$ are the vectors of state variables (unknown), $z_s(t) \in \mathbb{R}^m$ and $z_c(t) \in \mathbb{R}^l$ denote the output and input vectors (known variables). $z_c(t)$ may be equal to 0 in case of an uncontrolled system. The functions h, l, and g are linear or nonlinear functions that involve a set of parameters denoted $\mathcal{Z}_p$ (as tank diameter, nominal flow, ...).

For any vector ν, let us define $\bar{\nu}$ to stand for ν and its time derivatives up to some (unspecified) order.

▶ **Definition 4** (Analytical Redundancy Relations (ARR)). *ARRs are relations $\mathcal{M}'(\bar{z}) = \mathcal{M}''(\bar{f})$ obtained from $\mathcal{M}(z, x, f)$ by formally eliminating unknown variables x. While $\mathcal{M}''(\bar{f})$ is the internal form that depends on the faults and is not known, $\mathcal{M}'(\bar{z})$ is the computation form and can be computed from the known variables and their derivatives.*

$\mathcal{M}'(\bar{z})$ defines a set of ARRs. A single ARR takes the form $arr_i(\bar{z}') = r_i$, where r_i is a scalar signal named *residual* and $\bar{z}'$ a subvector of $\bar{z}$. It can be used as residual generator.

▶ **Definition 5** (Residual generator for $\mathcal{M}(z, x, f)$). *A relation of the form $arr_i(\bar{z}') = r_i$, with input $\bar{z}'$ a subvector of $\bar{z}$ and output r_i, a scalar signal named* residual*, is a residual generator for the model $\mathcal{M}(z, x, f)$ if, for all z consistent with $\mathcal{M}(z, x, f)$, it holds that $\lim_{t\to\infty} r(t) = 0$.*

In simpler terms, a residual generator produces a signal (residual) that should be zero when the system is working correctly, and any deviation from zero indicates a potential fault.

From the system model, ARRs can be obtained based on the analytical redundancy embedded in the model. For this, variable elimination can be applied, thereby obtaining relations that involve only known variables. ARRs allow us to assess whether the measurements z are consistent with the model $\mathcal{M}(z, x, f)$, hence defining *diagnosis tests*. Once a fault exists within the CPS it will hopefully have some influence on the measurements and therefore also on the residuals. If it does not have any detectable influence then the fault is said non detectable and it cannot be detected [15].

Following the ideas from Cassar and Staroswiecki [4] and Travé-Massuyès et al. [27], structural analysis can be advantageously used to obtain ARRs. It consists in abstracting the system model by keeping only the links between equations and variables. The main advantages are that it can be applied to large scale systems, linear or non linear, even under uncertainty.

When used for fault diagnosis purposes, structural analysis allows one to find subsets of equations endowed with redundancy. Structural redundancy $\rho_{\mathcal{M}'}$ of a set of equations $\mathcal{M}' \subseteq \mathcal{M}$ is defined as the difference between the number of equations and the number of unknown variables.

Actually, minimal subsets of equations endowed with structural redundancy have been proved to provide sets of equations supporting diagnostic tests [15]. These have been defined as Fault-Driven Minimal Structurally Overdetermined (FMSO) sets [22]. Assume $\mathcal{F}_\varphi$ as the set of faults that are involved in a set of equations $\varphi \subseteq \mathcal{M}(z, x, \mathtt{f})$.

▶ **Definition 6** (FMSO set). *A subset of equations $\varphi \subseteq \mathcal{M}(z, x, \mathtt{f})$ is an FMSO set of $\mathcal{M}(z, x, \mathtt{f})$ if (1) $\mathcal{F}_\varphi \neq \emptyset$ and $\rho_\varphi = 1$, (2) no subset of φ is overdetermined, i.e. with more equations than unknown variables. The set of FMSO sets of $\mathcal{M}$ is denoted Φ.*

FMSO sets can be converted into ARRs. By their nature, all the undetermined variables involved in an FMSO set φ can be resolved using $|\varphi| - 1$ equations. These variables can subsequently be substituted into the $|\varphi|^{th}$ equation to formulate an ARR off-line, which is then utilized on-line as a diagnostic test. Furthermore, the concept of an FMSO set is crucial for defining detectable faults and isolable faults. Here, we revisit these definitions [15].

▶ **Definition 7** (Detectable fault). *A fault $f \in \mathcal{F}$ is detectable in the system $\mathcal{M}(z, x, \mathtt{f})$ if there exists an FMSO set $\varphi \in \Phi$ such that $f \in \mathcal{F}_\varphi$.*

▶ **Definition 8** (Isolable faults). *Given two detectable faults f and f' of $\mathcal{F}$, $f \neq f'$, f is isolable from f' if there exists an FMSO set $\varphi \in \Phi$ such that $f \in \mathcal{F}_\varphi$ and $f' \notin \mathcal{F}_\varphi$.*

The Fault Diagnosis Toolbox (FDT) [10] is designed for the analysis and creation of fault diagnosis systems for dynamic systems, which are mainly characterized by differential-algebraic equations. Utilizing a structural model, this toolbox facilitates the production of FMSO sets. From these sets, it can automatically produce ARRs that are employed as diagnostic tests.

To summarise, in this article we are using a MLLM to generate system models. Then we use the above theory and employ the FDT to create ARRs and residuals for fault diagnosis.

3 Creating Physical Models for Fault Diagnosis Using MLLMs

This section presents a novel approach to create physical system models for fault diagnosis using a five-step approach to query a MLLM. Our goal is to obtain a physical model $\mathcal{M}$ describing a cyber-physical system's normal behaviour put in a form $\tilde{\mathcal{M}}$ which can be used as input for the FDT [10]. Using our model with the FDT we want to show how to automatically diagnose faults using the theory described in the section above. In particular, we solve $\mathcal{P}, \mathcal{I} \rightarrow \tilde{\mathcal{M}}$, with a P&ID $\mathcal{P}$, some additional external information $\mathcal{I}$, and create the physical model in a format suitable for the FDT $\tilde{\mathcal{M}}$.

Our goal is to obtain a method that practitioners can use to quickly create new system models without much expert knowledge and to test the capabilities of modern MLLMs in the domain of fault diagnosis. To realise this, we are relying mainly on the design of a system prompt of MLLMs, which we attempted to keep as generic as possible. In the user prompt, information specific to the system being modeled is provided. We also make use of the FDT, which is an established technique to generate ARRs for fault diagnosis from physical models. We will therefore show that our method can automatically create physical fault diagnosis models for process industrial use cases.

In every step, a prompt with some external information is sent to the MLLM that can be summarised as the tuple $(\mathcal{P}, \mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des})$, with P&ID $\mathcal{P}$, assumptions for the physical model $\mathcal{A}$, parameter names $\mathcal{Z}_p$, control variable names $\mathcal{Z}_c$, unknown variable names $\mathcal{X}$, and faults that should be detected, with a short description, $\mathcal{F}_{des}$. We refer to $\mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des} \in \mathcal{I}$ with external information $\mathcal{I}$.

The five steps of our prompting approach are the following: i) Let the MLLM read the diagram image data and represent it in some partially specified intermediate format, ii) Identify the sensors from the diagram, iii) Create the physical equations, iv) Match sensors and variables, and v) Format the model for the FDT. The steps are formalised in Algorithm 1 and will be described in more detail below. All steps provide the MLLM with the P&ID as input. In addition, the steps take a system message and a user prompt as input. The system message contains the generic task for the respective step. However, taking current abilities of MLLMs into account, it is still domain dependent. The user message contains a subset of the external information $\mathcal{I}' \subset \mathcal{I}$ which is specific for the system. The prompts can be found in GitHub[1]. So far, our approach aims to work for water tank systems with standard components, such as tanks, pumps, valves, flow indicators, and level indicators.

Algorithm 1 `Create Physical Models with MLLMs.`

Data: $\mathcal{P}, \mathcal{A}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{X}, \mathcal{F}_{des}$
Result: $\tilde{\mathcal{M}}$
1 $CompsConnections \leftarrow ReadDiagram(\mathcal{P})$;
2 $Sensors \leftarrow IdentifySensors(\mathcal{P}, CompsConnections)$;
3 $Equations \leftarrow CreateEquations(\mathcal{P}, CompsConnections, \mathcal{X}, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{A})$;
4 $\mathcal{M} \leftarrow Matching(\mathcal{P}, Sensors, Equations, CompsConnections, \mathcal{Z}_p, \mathcal{Z}_c, \mathcal{F}_{des})$;
5 $\tilde{\mathcal{M}} \leftarrow Formatting(\mathcal{P}, \mathcal{M})$
6 *return* $\tilde{\mathcal{M}}$;

Step 1: Read Diagram. In the first step, the components of the system and their connections are extracted from the diagram $\mathcal{P}$ with the vision capabilities of the MLLM. The output, which is formatted as a table with additional explanatory information is captured in *CompsConnections*. The model receives context about the diagram in the system message. It is requested to output the components and their connections in a simple table that can easily be interpreted by the MLLM in later steps. To avoid that the sensors are listed as elements, they should be ignored in this step. The model is also instructed to write 'unclear' for connections it cannot identify. In previous work we found that including these 'unclear' connections significantly reduces hallucinations [17]. At the end, the MLLM is told to make sure that the valves only occur once in the input and output columns. Otherwise, inconsistent models are generated that cannot be used by subsequent steps. In this step, no user prompt is needed, the input is just the diagram.

Step 2: Identify Sensors. The second step creates another table *Sensors* containing the existing sensors $\mathcal{Z}_s$, their type, and their placement from diagram $\mathcal{P}$ and from the output from step 1 in the form of *CompsConnections*. Context about P&IDs, possibly occurring

[1] `https://github.com/silkeme/DX24_Model_Creation_LLMs`
The repository contains the prompts, all mentioned information about the systems used for the evaluation, the resulting physical models, and all intermediate outputs.

sensors, and the placement of the sensors in the diagram are provided in the system message. To reduce hallucinations, the MLLM is advised to check if the combination of number and sensor type is actually in the image and that each number can only occur once. To make sure there is an output, the MLLM is advised to try the task, even if it tends to say the diagram is too complex. The user prompt contains only the input from the previous step *CompsConnections*.

Step 3: Create Equations. For the creation of the physical equations *Equations*, the MLLM gets the diagram $\mathcal{P}$, the names of the unknown variables $\mathcal{X}$, *CompsConnections*, the parameter names $\mathcal{Z}_p$, the control variable names $\mathcal{Z}_c$, and the assumptions $\mathcal{A}$ as input. The model is instructed to use as many of the given symbols as possible to take care that the equations are detailed enough for the later steps. It is also told to calculate the values that could be measured with the sensors (volume flow through valves and pumps, and levels of the tanks) to make the model suitable for fault diagnosis later. The 'unclear' connections from *CompsConnections* are excluded to obtain an executable model. In our approach, we prefer an incomplete model over a wrong model. At the end of the system message, there are some formatting instructions to prepare the equations for the usage in the FDT. Unlike in our previous work [17], we do not provide any equations and rely completely on the equations known to the MLLM to increase usability. In the user prompt, we include the external information $\mathcal{I}'$. To provide the model with additional context it is exemplary mentioned that the variables and parameters have the same names as in OpenModelica. When the method is applied to other systems that are not modelled with OpenModelica, the variables and parameters should have self-explaining names such that the MLLM is able to use them correctly. In this case, the hint to OpenModelica in the user prompt would need to be adjusted accordingly.

Step 4: Sensor Matching and Variable Assignment. In this step, all the results from the previous steps are merged, the faults are added to the equations and the resulting physical model $\mathcal{M}$ is created. The step takes as input the diagram $\mathcal{P}$, the identified *Sensors*, the *Equations*, *CompsConnections*, the parameter names $\mathcal{Z}_p$, the control variable names $\mathcal{Z}_c$, and the faults with a short description $\mathcal{F}_{des}$. The system message contains an explanation of how to handle the input, how to match the sensors $\mathcal{Z}_s$ with the variables from the equations, to identify parameters $\mathcal{Z}_p$, and how to handle control variables $\mathcal{Z}_c$. In addition, it is instructed to add the faults $\mathcal{F}_{des}$ as multiplicative or additive faults to the model. The variables should be stored in a dictionary, assigning them to the suitable key depending on if they are known ($\mathcal{Z}$), unknown ($\mathcal{X}$), faults ($\mathcal{F}$), or parameters. The user prompt contains only the mentioned input with a short introductive description.

Step 5: Format Model for Fault Diagnosis Toolbox. In the final step, the physical model $\mathcal{M}$ is transformed to be suitable for the Fault Diagnosis Toolbox into $\tilde{\mathcal{M}}$. In the system message, the structure of the model is provided which just needs to be filled in by the MLLM. It contains many specific instructions to how the format should look like and to force the MLLM to output only the desired format, since otherwise, $\tilde{\mathcal{M}}$ is not executable. The user prompt only contains the physical model $\mathcal{M}$ that already includes all the other external information at this point.

4 Evaluation

To evaluate our approach we used three different multiple tank systems (i.e. S1-S3) from the benchmark from Balzereit et al. [3]. The P&ID for system S1 is shown exemplarily in Figure 1. We manually re-created the P&IDs to ensure that the MLLM has not seen them before as part of its training data. The P&IDs follow the standard DIN EN ISO 10628 [6]. For each system we generated physical models to evaluate our MLLM approach. For the qualitative evaluation, we compared the models created by the MLLM to manually created models. We looked at the following aspects: visual analysis of the diagram, usability of the model, and correctness of the model. In addition, we performed a quantitative evaluation to check how our approach is able to diagnose real injected faults with simulated data.

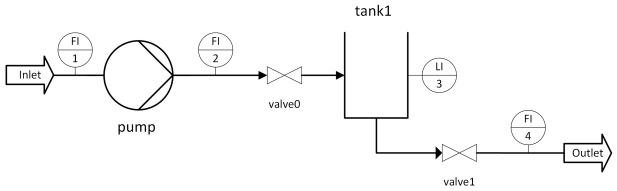


Figure 1 P&ID diagram of the one-tank system S1.

The implementation was done in Python, using the OpenAI library to call OpenAI's GPT4 via the API. The version of the model is 'gpt-4-vision-preview' and the following parameters were used to make the model less creative and more reliable: seed=42, temperature=0, top_p=0.1, frequency_penalty=0, and presence_penalty=0. We repeated the experiment for each of the three systems 100 times. For each step's input we used exactly the output of the previous step.

The data we used was created using OpenModelica 1.13 with the benchmark of Balzereit et al. [3]. We extracted the variable names and excluded internal variables (those starting with '$cse'). We listed the names of control variables $\mathcal{Z}_c$ separately and created another list with the names of the parameters $\mathcal{Z}_p$. A list with faults to be detected $\mathcal{F}_{des}$ was created manually for the systems. The assumptions $\mathcal{A}$ are used to simplify the model and are the same for all systems. An excerpt of external information $\mathcal{I}$ for system S1 is shown in Table 1. An overview of the number of elements and names is listed in Table 2.

Table 1 Inputs for S1.

Input	Count	Content
Parameter names $\mathcal{Z}_p$	16	['pipe_Diameter', 'pipe4_Diameter', 'tank1_Diameter', 'pipe1_Diameter', 'pipe2_Diameter', 'pipe_Length', ...]
Control variable names $\mathcal{Z}_c$	3	['pump_N', 'valve0_opening', 'valve1_opening']
Unknown variable names $\mathcal{X}_n$	348	['time', 'pump_medium_T', 'pump_medium_p', 'tank1_level', 'tank1_medium_T', 'der_pump_medium_T', 'der_pump_medium_p', ...]
Faults $\mathcal{F}_{des}$	4	Leakage of tank1: 'f_tank1leak'. Valve0 blocked: 'f_valve0'. Valve1 blocked: 'f_valve1'. Degraded rotational speed of pump: 'f_pumpSlow'.
Assumptions $\mathcal{A}$	5	\nThe fluid in the system is water\nThe fluid is incompressible\nThere are no energy losses\nThe process is adiabatic\nThe tanks are open.

Table 2 Overview of the number of elements and names in S1-S3.

System	Components	Sensors	$\mathcal{Z}_p$	$\mathcal{Z}_c$	$\mathcal{X}_n$	$\mathbf{F}_{des}$	$\mathcal{A}$
S1	4	4	16	3	348	4	5
S2	8	7	32	5	691	5	5
S3	11	9	36	7	1007	4	5

4.1 Qualitative Evaluation

Our motivation for the qualitative evaluation was to generate insights on how well current MLLMs can be used for fault diagnosis. We therefore manually analyzed the models created for each system. The models were assessed according to the following criteria: i) Visual analysis of the diagram. We investigated the recognition of elements, of connections, and sensors (step 1 and step 2). ii) Usability of the model. We identified errors that lead to non-executable models and analysed how this affects usability. iii) Correctness of the model. We checked the assignment of the variables to the four categories (unknown variables $\mathcal{X}$, faults $\mathcal{F}$, known variables including control variables $\mathcal{Z}_c$ and sensor variables $\mathcal{Z}_s$, and parameters $\mathcal{Z}_p$), the number and design of equations, and if the assumptions were interpreted correctly. We will now present each of the analysis results in detail.

i) Visual Analysis of the Diagram. In step 1 (Read Diagram) almost all components were recognized correctly, except for some cases in S3, where *valveLinear7* was missing. The sensors were successfully ignored in this step, such that they are not misclassified as components. The complete structure of S1 and S2 was identified correctly in all cases. For S3 with its complex structure and more elements, some connections were hallucinated or confused. The upper part of the diagram was always recognized correctly, while the connections to valveLinear7 were never identified and the output of valveLinear5 was interpreted diversely. This leads to wrong inputs for tank4 and, in some cases, valveLinear6. Some repetitions were completely correct, except for valveLinear7, which was marked as unclear. A model resulting from that would be incomplete and not wrong. In step 2 (Identify Sensors) the level indicators were nearly always correct in all systems. The flow indicators were always correct in S1. In S2, FI3 was always wrong, FI7 was sometimes right. The sensors were interpreted left of valveLinear1, and valveLinear3, respectively, instead of the actual placement at the right side of the valves. The wrong placement does not necessarily make the model wrong in the end, if the sensor is assigned to the valve and there is no leak in the pipe. The same issue occurred in S3, where in addition two sensors (FI15 and FI16) were identified at wrong locations, such as between valveLinear3 and tank3 for sensor FI15. Also in the mostly correct repetitions, these two sensors were at the wrong position, leading to wrong measurement assignments. In total, there was not much variation in the outputs of steps 1 and 2, the same tables occurred quite often. For a more detailed evaluation on the frequency of unique results, please check our previous work [17].

ii) Usability of the Model. We validated the output of step 3-5 by checking the final model for its usability from an expert's standpoint. In all repetitions for all systems, the models had the correct basic structure, consisting of the import of the required libraries and the dictionary in which the model was stored. The correct variable names were used in most cases but sometimes variables were not defined or contained special characters which cannot be handled by the Diagnosis Toolbox, leading to non-executable models. ChatGPT was

advised to check if variables are in the equations but not in the dictionary with the variables. Instead of adding the missing variables to the dictionary directly, they were added at the end of the model in some cases. The missing symbol definition resulted in a non-executable model, if the symbol was needed for solving the equations. It did the same for equations, but since they do not need to be defined as symbols, the models were still executable. Some model definitions being not exactly consistent with the modeling conventions of the Diagnosis Toolbox, for example that only variables in $\mathcal{X}$ can be used in differential constraints, often result in index out of bound errors during matching computation while creating the residuals. For residuals, the values for some parameters, such as π or the gravity acceleration, were not present since they were not defined in the model provided by ChatGPT. Some of the successfully computed residuals did not work due to numerical errors, such as square root of a negative number, division by zero, or numerical overflow. An examination of how many residuals were calculated correctly, can be found in the quantitative evaluation.

In addition to the errors leading to non-executable models, we made some observations that did not disturb the usability of the model. The format of the models varies. Sometimes the equations were defined directly in the model structure, sometimes they were defined before and converted afterwards. Sometimes they were stored in variables that are summarized in a list later, sometimes they were directly listed. And sometimes single equations were added to the dictionary at a later point. In the majority of the models, the equations were suitably structured and the code was well commented, making it easy to gain an overview of the model.

iii) Correctness of the Model. The assignment of the given variables ($\mathcal{X}$, $\mathcal{F}$, $\mathcal{Z}_c$, $\mathcal{Z}_s$, and $\mathcal{Z}_p$) to the four categories unknown variables, faults, known variables, and parameters, was in most cases completely correct, and depended on the sensors that were identified in step 2. Faults and known variables were always correct, parameters sometimes contained values that were not in $\mathcal{Z}_p$. Sometimes, the unknown variables also contained some parameters, which is probably due to the instructions of step 4 (Sensor Matching) to store all not-assigned variables in the unknown variables. The consequence are more unknown variables and thus potentially an under-determined model for the system.

The number of equations varies strongly. Many models have one equation per component as intended, others have more redundant equations, such as equations for the water volume and mass in the tank in addition to the level. In general, there are many models with correct equations. In some models, single equations were wrong. In other models, it seems like more assumptions were applied than provided, resulting in simpler equations with fewer variables, such that, for example, the model works only with nominal flow. In general, the assumptions were included as intended, for example by ignoring the density of water. Sometimes the wrong variables were used, such as the pump's volume instead of the pump's volume flow. For each sensor there should be an extra equation but in some cases, the sensors were added to other equations as well. Sometimes the sensors were matched with the wrong variables, for example a level sensor was matched with the derivative tank level instead of the tank level. Under these conditions, residuals can be calculated without errors but do not work according to theoretical fault sensitivity, which will be checked in the quantitative evaluation. Another aspect are missing equations. ChatGPT does not add flow balance equations to valves or pumps, only to tanks. More faults might be detected with these equations. The system message of step 3 (Create Equations in Python) might be the reason why the balance equations are not there.

4.2 Quantitative Evaluation

The Quantitative evaluation was done with the Fault Diagnosis Toolbox [10]. Our five-step approach generated models in the form of a Python module that is expected to be consistent with FDT requirements. Only modules that load without errors are considered for further evaluation. We created a FDT model object for each model, computed all FMSO sets [15], and selected only the sets that can give residuals in integral or algebraic causality. Integral causality is preferred to derivative causality because numerical differentiation is noisy, and residual generators in integral causality correspond to the standard state-space formulation (eq. 1). For each FMSO, we tried to create a residual generator (corresponding to an ARR). We evaluated each correct residual generator on simulated benchmark data and verified fault detection and isolation performance. For each system S1-S3, we evaluated 100 generated models.

Table 3 presents the number of models imported into Python without errors and the mean number of FMSOs computed for each model. For each FMSO, we tried to compute residual generators. The following rows of Table 3 show the mean number of residual generators computed without errors (in the ideal case, we have one residual generator per FMSO) and the number of residuals that were correctly evaluated on the simulation files (in the ideal case each residual generator gives one residual that can be assessed with data). For S1, we consider tank leakage (tank1leak) and blockage of valve0. The last rows of Table 3 show the fraction of cases when these faults were successfully detected out of all correctly imported models. Fault detection results for S2 and S3 are presented in a similar way.

Additionally, Table 4 shows the frequencies of ambiguity groups. For S1, in 74% of cases, all system states are correctly isolated; in 12% of cases, only tank1 leak can be detected; in 10% of cases, only valve0 fault can be detected, and in 3% of cases, none of the faults is detected. For S2, in 73% of cases, none of the faults are detected; in 25% of cases, only tank2 leak can be detected, and in 2% of cases, all system states are correctly diagnosed. For S3, in 1% of cases, all faults can be detected; in 4% of cases, all faults except tank2 leak are correctly diagnosed, and in 34% of cases, none are detected. We assume that the errors in the image interpretation lead to the bad performance for S3. Table 4 also shows the statistically required number of executable models to expect at least one that supports the ambiguity group with a chance of 95%. The number of models n is calculated with

$$n \geq \frac{\ln(1-P)}{\ln(1-p)}, \tag{2}$$

where P is the desired chance (95%), and p is the frequency in which the ambiguity group was observed.

Figure 2 shows computed residuals for module S1_3, which gives complete fault isolability for two simulation files. In each file, the fault starts 250 seconds after the start of the simulation. We can observe that residual generator ResGen0 is sensitive to tank leakage, and residuals ResGen2 and ResGen4 are sensitive to valve blockage. ResGen0 computes tank level from the flow measurements and equation e_1; ResGen2 and ResGen4 use valve equations. It can be observed that these residuals are not precisely zero on the time intervals where they should, for instance ResGen2 and ResGen4 should be zero in [t=0,t=250], as the approximate valve flow is based on nominal flow and valve opening. As valve differential pressure measurement is not available, this is a reasonable attempt.

For comparison, we created the model for S1 by hand (see GitHub). It correctly detects and isolates two faults considered for S1 (residuals in Fig. 3). Therefore, we achieve optimal performance in 74% of automatically generated models and would need 3 models to have a probability of at least 95% that one of the models has optimal performance (Table 4).

Table 3 Model results of models generated for S1-S3. Valve faults are cloggings and tank faults are leaks.

System	S1		S2		S3	
# correctly imported models	93		85		83	
Mean # FMSOs	5.40		22.27		48.70	
Mean # corr. res. gen. per FMSO	0.9830		0.2247		0.5783	
Mean # correct res. per gen.	0.9825		1		0.9251	
Individual Fault Isolation Accuracy	valve0	0.8495	tank1	0.0235	pipe4	0.2169
	tank1	0.8710	tank2	0.2706	tank2	0.0120
					valve3	0.6627
					valve6	0.1928

For S2 and S3 we found a common pattern for worse performing cases. We observe that the generated models do not adequately handle the situations where the flow is split (like after pump in S2) and where the flows merge (like before tank4 in S3). The lack of direct measurements of tank inflows causes poor performance for tank1 leak in S2. The poor performance for tank2 leak in S3 is caused by the lack of measurement of tank4 inflow. These situations can be correctly detected with correct models but require writing balance equations for flows.

Table 4 Ambiguity groups for S1 and S3 with the required sample size (indicating how many executable models need to be created to have at least one that works for the ambiguity group with a probability of 95%).

System	Ambiguity groups	Frequency	Required Sample Size
S1	{valve0}, {tank1leak}, {NF}	0.7419	3
	{valve0, NF}, {tank1leak}	0.1183	24
	{valve0}, {NF, tank1leak}	0.0968	30
	{valve0, tank1leak, NF}	0.0323	92
	{valve0, tank1leak}, {NF}	0.0108	276
S2	{tank2leak, NF, tank1leak}	0.7294	3
	{tank2leak}, {NF, tank1leak}	0.2471	11
	{tank2leak}, {NF}, {tank1leak}	0.0235	126
S3	{valve6, tank2leak, pipe4, NF}, {valve3}	0.4458	6
	{pipe4, valve6, valve3, NF, tank2leak}	0.3373	8
	{pipe4, valve6}, {NF, tank2leak}, {valve3}	0.1325	22
	{pipe4}, {NF, tank2leak}, {valve3}, {valve6}	0.0482	61
	{pipe4}, {valve3}, {valve6, tank2leak, NF}	0.0241	123
	{pipe4, valve6, tank2leak, valve3}, {NF}	0.0120	249

5 Discussion

The evaluation shows the potential of MLLMs like OpenAI's GPT4 (ChatGPT) for the fully automated creation of models for fault diagnosis. Within the five steps of our approach, there are many influencing factors and many decisions to be made by the MLLM that are potentially wrong. Especially because MLLMs are highly non-deterministic. Under these circumstances, we find the results to be surprisingly good. Even though the quantitative

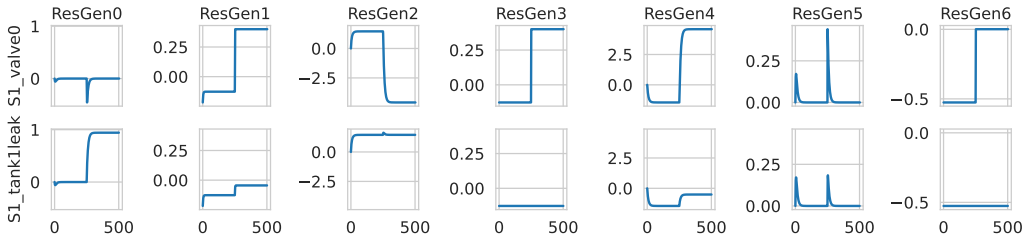


Figure 2 Residuals for model S1_3.

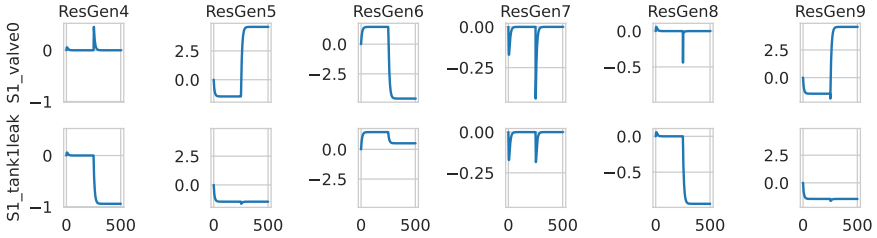


Figure 3 Residuals for the correct model.

evaluation shows severe limitations of overall fault diagnosis performance, it indicates that the creation of such models is possible with MLLMs, especially to support practitioners with some acceptable drafts they can choose from. By creating a number of models with our approach and evaluating them in terms of executability and correctness, a good model can be selected. However, in its current form, our approach is not reliable enough to be applied without an expert checking the models.

We evaluated our approach with simulated data from OpenModelica. Hence we do not know how it reacts to other sources of data, such as real-world data. In S3, some of the faults were easier to detect than tank1 leak in S2, which makes the results not fully comparable between the systems. In our opinion, the image analysis is the biggest issue, as those diagrams are the most common in the process industry. While the image analysis works perfectly for small systems, the missing ability to detect long lines and the correct placement of sensors for complex systems leads to errors in the first two steps which then subsequently propagate. With sensors assigned to the wrong variables, it is not possible to calculate correct residuals, even though the residual generators were created successfully. A more reliable method to extract the system structure needs to be developed. Alternatively, the extraction of the structure and sensor placements could be done manually and provided to the model as input for step 3. We assume, that the approach does not work well for systems in which the flow is split, since ChatGPT did not create flow balance equations correctly. This issue might be solved by explicitly mentioning it in the prompt of step 3. Another way to improve the results could be to include sample equations to reduce the arbitrariness of the models and to be less dependent on the data the MLLM has seen during training. This was done in our previous work [17] successfully but has the disadvantage that suitable equations need to be identified in advance. Our approach was only validated on ChatGPT so far which makes it unpredictable how it might work with other MLLMs.

Unfortunately we had to restrict our approach within the prompts with the following assumptions. Our approach is intended to only work for flow sensors that are between exactly two elements, valves that are connected to one element without a split or a merge, components and sensors with unique names, and P&IDs that follow the standard DIN EN

ISO 10628 [6]. The units of the variables and parameters were not provided to ChatGPT and could lead to wrong residuals if they are not consistent. Since MLLMs are inherently non-deterministic, the reliability of our quantitative evaluation based on 100 repetitions for each system is uncertain.

6 Conclusion

We presented a novel five step-approach to generate physical models for fault diagnosis with MLLMs. We showed that it is possible to generate suitable models with a completely automated chain for small systems. In 74% of the cases we were able to detect and isolate all faults for the small system S1. In 97% of the cases, we could detect and isolate at least one fault correctly. For the other two systems, the performance was weaker, leading to a detection rate of 27% for S2 and 66% for S3. For S3 in no case the faults were completely isolable. Our results show that MLLMs can be used for the generation of models for fault diagnosis, but so far MLLMs are not reliable enough to truly automate model creation. Instead, we still need an expert to check and evaluate the generated models. The main weakness we identified is that the bad performance mainly stems from the imprecise detection of connections between elements and sensor placements (i.e. the image interpretation by the MLLM). However, we think that the image recognition feature of MLLMs will strongly improve in the future and scalable divide-and-conquer approaches will emerge. Future work should focus on a more reliable method to extract the system structure of complex systems, evaluating the approach with other MLLMs, and trying more assumptions about the system behaviour,

References

1 Luis A Aguirre. An introduction to nonlinear system identification. In *Lectures on Nonlinear Dynamics*, pages 133–154. Springer, 2023.

2 Lukas Schulze Balhorn, Marc Caballero, and Artur M Schweidtmann. Toward autocorrection of chemical process flowsheets using large language models. *arXiv preprint arXiv:2312.02873*, 2023. doi:10.48550/arXiv.2312.02873.

3 Kaja Balzereit, Alexander Diedrich, Jonas Ginster, Stefan Windmann, and Oliver Niggemann. An ensemble of benchmarks for the evaluation of ai methods for fault handling in cpps. In *19th IEEE International Conference on Industrial Informatics*, November 2021.

4 J-Ph Cassar and M Staroswiecki. A structural approach for the design of failure detection and identification systems. *IFAC Proceedings Volumes*, 30(6):841–846, 1997.

5 Cody James Christopher and Alban Grastien. Critical observations in model-based diagnosis. *Artificial Intelligence*, page 104116, 2024. doi:10.1016/J.ARTINT.2024.104116.

6 Deutsches Institut für Normung e.V. (DIN). DIN EN ISO 10628-2: Flow diagrams for process plants - part 2: Graphical symbols. DIN standard, DIN, 2012.

7 Alexander Diedrich, Lukas Moddemann, and Oliver Niggemann. Learning system descriptions for cyber-physical systems. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

8 Marius-Constantin Dinu, Claudiu Leoveanu-Condrei, Markus Holzleitner, Werner Zellinger, and Sepp Hochreiter. Symbolicai: A framework for logic-based approaches combining generative models and solvers. *arXiv preprint arXiv:2402.00854*, 2024. doi:10.48550/arXiv.2402.00854.

9 Teresa Escobet, Anibal Bregon, Belarmino Pulido, and Vicenç Puig. *Fault Diagnosis of Dynamic Systems*. Springer, 2019.

10 Erik Frisk, Mattias Krysander, and Daniel Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *IFAC-PapersOnLine*, 50(1):3287–3293, 2017.

11 Constantin Hildebrandt, Sebastian Törsleff, Birte Caesar, and Alexander Fay. Ontology building for cyber-physical systems: A domain expert-centric approach. In *2018 IEEE 14th international conference on automation science and engineering (CASE)*, pages 1079–1086. IEEE, 2018. doi:10.1109/COASE.2018.8560465.

12 Edwin Hirtreiter, Lukas Schulze Balhorn, and Artur M Schweidtmann. Toward automatic generation of control structures for process flow diagrams with large language models. *AIChE Journal*, 70(1):e18259, 2024.

13 Sungmin Kang, Gabin An, and Shin Yoo. A preliminary evaluation of llm-based fault localization. *arXiv preprint arXiv:2308.05487*, 2023. doi:10.48550/arXiv.2308.05487.

14 Shota Kato, Chunpu Zhang, and Manabu Kano. Simple algorithm for judging equivalence of differential-algebraic equation systems. *Scientific reports*, 13(1):11534, 2023.

15 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(1):197–206, 2008. doi:10.1109/TSMCA.2007.909555.

16 Shouvik Mani, Michael A. Haddad, Dan Constantini, Willy Douhard, Qiwei Li, and Louis Poirier. Automatic Digitization of Engineering Diagrams using Deep Learning and Graph Search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 673–679. IEEE, June 2020. doi:10.1109/CVPRW50498.2020.00096.

17 Silke Merkelbach, Alexander Diedrich, Sebastian Von Enzberg, Oliver Niggemann, and Roman Dumitrescu. Towards the generation of models for fault diagnosis of cps using vqa models. *ML4CPS 2024 – Machine Learning for Cyber Physical Systems Conference*, 2014.

18 Lukas Moddemann, Henrik Sebastian Steude, Alexander Diedrich, and Oliver Niggemann. Discret2di - deep learning based discretization for model-based diagnosis. In *Proceedings of 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 2024.

19 Oluwatosin Ogundare, Gustavo Quiros Araya, Ioannis Akrotirianakis, and Ankit Shukla. Resiliency analysis of llm generated models for industrial automation. *arXiv preprint arXiv:2308.12129*, 2023. doi:10.48550/arXiv.2308.12129.

20 OpenAI. Chatgpt 4 vision preview. version gpt-4-1106-vision-preview. url: https://platform.openai.com/docs/api-reference, 2023.

21 LIU Peifeng, Lu Qian, Xingwei Zhao, and Bo Tao. Joint knowledge graph and large language model for fault diagnosis and its application in aviation assembly. *IEEE Transactions on Industrial Informatics*, 2024.

22 CG Pérez-Zuniga, E Chanthery, L Travé-Massuyès, and J Sotomayor. Fault-driven structural diagnosis approach in a distributed context. *IFAC-PapersOnLine*, 50(1):14254–14259, 2017.

23 Swantje Plambeck, Aaron Bracht, Nemanja Hranisavljevic, and Goerschwin Fey. Famos–fast model learning for hybrid cyber-physical systems using decision trees. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2024. doi:10.1145/3641513.3650131.

24 Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024. doi:10.48550/arXiv.2404.18400.

25 Arka Sinha, Johannes Bayer, and Syed Saqib Bukhari. Table localization and field value extraction in piping and instrumentation diagram images. In *2019 international conference on document analysis and recognition workshops (ICDARW)*, volume 1, pages 26–31. IEEE, 2019. doi:10.1109/ICDARW.2019.00010.

26 Louise Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27:1–16, 2014. doi:10.1016/J.ENGAPPAI.2013.09.018.

27 Louise Travé-Massuyes, Teresa Escobet, and Xavier Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 36(6):1146–1160, 2006. doi:10.1109/TSMCA.2006.878984.

28 Yonghao Wu, Zheng Li, Jie M Zhang, Mike Papadakis, Mark Harman, and Yong Liu. Large language models in fault localisation. *arXiv preprint arXiv:2308.15276*, 2023. doi:10.48550/arXiv.2308.15276.

Summary of "A Lazy Approach to Neural Numerical Planning with Control Parameters"

René Heesch
Helmut-Schmidt-University, Hamburg, Germany

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy

Jonas Ehrhardt
Helmut-Schmidt-University, Hamburg, Germany

Alexander Diedrich
Helmut-Schmidt-University, Hamburg, Germany

Oliver Niggemann
Helmut-Schmidt-University, Hamburg, Germany

Abstract

This is an extended abstract of the manuscript "A Lazy Approach to Neural Numerical Planning with Control Parameters" [2]. The paper presents a lazy, hierarchical approach to tackle the challenge of planning in complex numerical domains, where the effects of actions are influenced by control parameters, and may be described by neural networks.

2012 ACM Subject Classification Computing methodologies → Planning with abstraction and generalization

Keywords and phrases Satisfiability Modulo Theory, Neural Numerical Planning with Control Parameters, Neural Networks

Digital Object Identifier 10.4230/OASIcs.DX.2024.32

Category Extended Abstract

Funding This research as part of the projects EKI and LaiLa is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

Acknowledgements This work has benefitted from Dagstuhl Seminar 24031 "Fusing Causality, Reasoning, and Learning for Fault Management and Diagnosis."

1 Extended Abstract

This paper is an extended abstract of the manuscript "A Lazy Approach to Neural Numerical Planning with Control Parameters" that was published at the 27TH European Conference on Artificial Intelligence [2]. We refer the interested reader to that version for more details on our work.

Planning is essential for managing complex processes in real-world domains, such as factory automation, where actions must be sequenced and their control parameters be defined accordingly. While rule-based representations may struggle to model real world processes, neural networks (NNs) can approximate even complex dependencies, based on recorded or simulated data[1]. In the paper "A Lazy Approach to Neural Numerical Planning with Control Parameters" [2], we address the challenge of planning in complex numerical domains, where actions are indexed by control parameters, and their effects may be modeled using NNs [3]. We refer to this type of action as neural actions. Our work introduces the Lazy


© René Heesch, Alessandro Cimatti, Jonas Ehrhardt, Alexander Diedrich, and Oliver Niggemann;
licensed under Creative Commons License CC-BY 4.0



Neural Planner (LNP), a Satisfiability Modulo Theory (SMT) based planner, which allows for the integration of neural actions. LNP leverages on two core aspects: the abstraction of the neural actions on the SMT level and their subsequent concretization of control parameters.

First, an abstract plan is generated by an SMT solver, where NNs are treated as uninterpreted functions. This abstraction simplifies the planning problem by avoiding the complexity of direct reasoning about the internal structure of the NN and thus avoids the restriction to architectures that could be handled by SMT solvers. Once an abstract plan is found, this plan is concretized by querying the NNs to determine specific values for the control parameters. If a valid set of control parameters can be found, a final plan is found. If the concretization fails – meaning that no suitable parameters could be found – the system refines the abstraction by incorporating additional constraints, which are then lifted back to the SMT solver. The solver then continues searching for another abstract plan.

This lazy, hierarchical approach offers several advantages over existing methods. One key benefit is the flexibility to handle NNs as black-box models, where the internal details of the network, such as architecture and weights, are unknown or inaccessible. We differentiate three types of NN models in our approach: (i) full-access white-box models, where the network's details are fully known, (ii) function-access black-box models, which allow certain functionalities like backpropagation but do not make the entire model accessible, and (iii) no-access black-box models, where the network is treated purely as an input-output mapping. For full-access and function-access models, we apply a gradient-based optimization technique to find control parameters. For no-access models, we employ a search algorithm to explore the parameter space and identify a feasible solution.

Experimental evaluations on four different planning domains demonstrate the efficiency of the LNP approach. The results indicate that avoiding symbolic reasoning about NNs not only improves computational efficiency but also enables the integration of NNs as black-box models, expanding the applicability of numerical planning in domains where exact NN specifications are not available. Furthermore, we analyze the effects of integrating information from the training process of the neural actions. We show that integrating this information can reduce the runtime with the risk of increasing incompleteness.

While our LNP approach offers efficiency and generality compared to eager encodings, it trades off completeness due to the abstraction process. The approach is correct in that every plan found is valid, but it may not find all possible solutions. Nonetheless, LNP represents a significant advancement in the integration of NNs into planning problems, particularly for scenarios where explicit symbolic reasoning about NNs is either impractical or impossible.

In future work, we plan to explore a tighter integration of NN reasoning within the SMT solver, aiming at balancing the trade-offs between completeness and efficiency. We additionally intend to expand our approach to temporally extended planning problems and to investigate different optimization techniques for control parameter estimation in various NN access scenarios.

Our LNP approach enables automated planning in complex, real-world domains with control parameters. Moreover, it presents a method of integrating ML models into symbolic reasoning, which reduces the modeling effort and hence could enhance the capabilities of symbolic AI approaches in other applications, such as diagnosis.

References

1 Jonas Ehrhardt, René Heesch, and Oliver Niggemann. Learning process steps as dynamical systems for a sub-symbolic approach of process planning in cyber-physical production systems. In *Artificial Intelligence. ECAI 2023 International Workshops*, pages 332–345, Cham, 2024. Springer Nature Switzerland.

2 René Heesch, Alessandro Cimatti, Jonas Ehrhardt, Alexander Diedrich, and Oliver Niggemann. A lazy approach to neural numerical planning with control parameters. In *ECAI 2024*, pages 4262–4270. IOS Press, 2024. doi:10.3233/FAIA241000.

3 René Heesch, Jonas Ehrhardt, and Oliver Niggemann. Integrating machine learning into an SMT-based planning approach for production planning inc yber-physical production systems. In *Artificial Intelligence. ECAI 2023 International Workshops*, pages 318–331, Cham, 2024. Springer Nature Switzerland.

Summary of "Randomized Problem-Relaxation Solving for Over-Constrained Schedules"*

Patrick Rodler[1]
Department of Artificial Intelligence and Cybersecurity, University of Klagenfurt, Austria

Erich Teppan
Department of Artificial Intelligence and Cybersecurity, University of Klagenfurt, Austria

Dietmar Jannach
Department of Artificial Intelligence and Cybersecurity, University of Klagenfurt, Austria

Abstract

We present a general framework for tackling over-constrained job shop scheduling problems (JSSP) where the volume of jobs (orders) exceeds the production capacity for a given planning horizon. The goal is to process as many or as utile jobs as possible within the available time. The suggested framework approaches this optimization problem by solving multiple randomly modified relaxed problem instances, thereby taking a sample in a solution space that covers all optimal solutions. By continuously storing the best solution found so far, the result is a complete anytime algorithm that incrementally approximates an optimal solution. The proposed framework allows for highly parallel computations, and all of its modules are treated as black-boxes, allowing them to be instantiated with the most performant algorithms for the respective sub-problems. Using IBM's cutting-edge CP Optimizer suite, experiments on well-known JSSP benchmark problems demonstrate that the proposed framework consistently schedules more jobs in less computation time compared to a standalone constraint solver approach. Due to the generality of the proposed approach and its applicability to computing minimum-cardinality or most preferred minimal diagnoses, this work has the potential to positively impact the field of model-based diagnosis.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Computing methodologies → Randomized search; Computing methodologies → Optimization algorithms; Computing methodologies → Parallel algorithms; Computing methodologies → Modeling and simulation; Applied computing → Industry and manufacturing

Keywords and phrases diagnosis computation, randomized diagnosis computation, minimum-cardinality diagnoses, most preferred diagnoses, maximum-probability diagnoses, applications of diagnosis (over-constrained scheduling problems), diagnosis-based optimization, constraint programming, CP Optimizer, job shop scheduling problem, job set optimization problem, operations research, scheduling, industry use cases, minimal subset subject to a monotone predicate (MSMP) problem, problem relaxation, sampling for optimization, anytime algorithm

Digital Object Identifier 10.4230/OASIcs.DX.2024.33

Category Extended Abstract

Related Version *Full Version*: `https://doi.org/10.24963/kr.2021/72` [12]

Funding This work was supported by FFG, contract FO999910235.

[1] Corresponding author.
* See [12] for the Full Paper.



© Patrick Rodler, Erich Teppan, and Dietmar Jannach;
licensed under Creative Commons License CC-BY 4.0



Summary

Introduction and Motivation. Job scheduling is a crucial task in production industries. One important and frequently addressed problem is the *Job Shop Scheduling Problem (JSSP)* [2], which is NP-hard in the general case. Given a set of machines and a set of jobs, where each job consists of an ordered sequence of operations, with each operation having a specified duration and needing to be executed on a particular machine, the JSSP involves finding a schedule that assigns a start time for each operation on its respective machine such that *(i)* each operation of a job can only begin after the preceding operation of the same job is completed, *(ii)* on each machine, the next operation may only start once the current operation is finished, and *(iii)* the completion time (i.e., the time required to process all jobs) is minimized. The (NP-complete [4]) decision version of a JSSP involves a deadline as an additional input, and asks whether there is a schedule satisfying criteria (i) and (ii) with a completion time before the deadline.

Different methods have been applied to approach JSSPs. Among those, constraint programming (CP) has a long and successful history, and present-day CP solving systems are able to handle large-scale problem instances. However, today's often highly dynamic production regimes, supporting, e.g., make-to-order or lean production, lead to optimization problems on top of the underlying JSSPs which may significantly increase computation times.

One typical problem of that type arises when the set of orders (jobs) exceeds the current production capacities with respect to a given planning horizon (e.g., a week). In such a situation the producer is confronted with an over-constrained JSSP, i.e., it is not possible to work off all the orders in time and it must be decided which of the jobs to postpone. We call the task of finding a job set of maximal utility (e.g., revenue) that can be finished within a given planning horizon the *Job Set Optimization Problem (JOP)*, which is NP-hard [1].

When using constraint programming, a straightforward approach to solving JOPs is to adapt the CP encoding of the over-constrained JSSP by adding an optimization statement, which effectuates that the utility of jobs scheduled and finished prior to a given deadline is maximized. We call this the *direct CP approach* to JOP. Since however the JOP represents a hard problem on top of the JSSP, also the most powerful state-of-the-art CP solvers may struggle with the increased problem complexity.

Approach. As a remedy to this, we propose a framework to tackle the JOP based on the observation that the problem of computing a subset-minimal job set to be postponed *(i)* is a relaxation of the JOP, *(ii)* can be solved using a linear number of CP solver calls, each for one JSSP decision problem, and *(iii)* is not directly supported by current CP solvers. The idea is to compute multiple such subset-minimal job sets in a random way, thus intuitively taking a random sample in a solution space that covers all JOP solutions. By always storing the best found solution, the framework allows to successively approach a JOP solution.

Compared to the direct CP approach, which can be seen as tackling two problems at once, i.e., the (implicit) subset-minimality as well as the optimal utility of the JOP solution, our framework achieves a disentanglement of these two problems by extracting the (efficient and well understood) MSMP [5] (*M*inimal *S*ubset subject to a *M*onotone *P*redicate) reasoning from the solver and leaving to the solver the role of deciding a polynomial number of JSSP instances (for which state-of-the-art solvers are optimized).

Our framework consists of three modules: a *CP solver* for decision versions of JSSP, a sufficiently general [7] *MSMP algorithm* for finding subset-minimal job sets, and a *random number generator* for creating multiple random subset-minimal job sets to enable optimization.

Remarks: *(Anytime Property)* Our approach is an anytime algorithm, i.e., the solution quality increases monotonically throughout the solving process. *(Completeness)* Given sufficient time and a suitable full-cycle random number generator (cf. [13]), the algorithm will always produce a JOP solution. *(Adaptability)* Each module in the framework is viewed as a black-box, allowing integration of different algorithms and benefiting from the latest research in JSSP and MSMP. *(Parallelization)* No information exchange is required between iterations, enabling efficient multi-threaded implementations of our approach. *(Automation)* No manual adaptation of the CP encoding of a given JSSP instance to a JOP encoding is needed. *(Generality)* The approach is applicable to computing minimum-cardinality or most preferred diagnoses for any model-based diagnosis problem as per Reiter's definition [6] (see below).

Evaluation. To compare the proposed framework with a direct CP approach, we conducted evaluations on 100 JOP instances extracted from Taillard's well-known JSSP benchmark suite [15] (where uniform job utility is assumed). We implemented the direct CP approach by IBM's CP Optimizer, a currently leading CP solver [3]; in our framework, we instantiated the CP solver also by CP Optimizer for a fair comparison, the MSMP algorithm by the random input permutation approach for Inv-QX [14] proposed by [9], and the random number generator by the respective Java function. Our results show that the use of the proposed framework *always leads to schedules involving more in-time finished jobs* than the direct CP approach. Specifically, our approach schedules an avg. of 8 % and up to 15 % more jobs for instances with 50 jobs and 15 machines, and 5 % (avg.) and 13 % (max.) more for instances with 100 jobs and 20 machines. This means a reduction in the number of postponed jobs (cf. diagnosis cardinality in model-based diagnosis) by 27 % (avg.) and 63 % (max.). Additionally, our approach is *always significantly faster* as it constantly yields better results (up to 12 % more scheduled jobs) within 1 hour than the direct CP approach achieves in 2 hours.

Relation to Model-Based Diagnosis. The proposed framework to approach the JOP is generally applicable to compute most preferred (or minimum-cardinality) diagnoses for any diagnosis problem $\mathcal{P} := (\text{SD}, \text{COMPS}, \text{OBS})$ as per Reiter's theory [6]. This can be recognized by observing that *(1)* the computation of a minimal diagnosis given $\mathcal{P}$ is an MSMP problem (cf. [7]), just as the computation of a subset-minimal job set that needs to be postponed to satisfy a scheduling deadline is an MSMP problem, and *(2)* the problem of computing a minimal diagnosis is a relaxation of the problem of computing a most preferred (or minimum-cardinality) diagnosis[2], just as finding a subset-minimal job set to be postponed is a relaxation of the JOP. Hence, put in diagnostic terms, our approach involves the computation of a random set of minimal diagnoses for a given diagnosis problem $\mathcal{P}$ while always storing the best found diagnosis so far, aiming to ultimately find one or more most preferred (or minimum-cardinality) diagnoses. According to the classification scheme suggested in [10] (see the definitions there), the diagnosis algorithm based on our approach is sound, complete, not best-first, multiple-solution, not conflict dependent, generally applicable, logics-agnostic, and space-efficient, and it uses black-box reasoning.

In fact, as discussed in [11], the JOP can be interpreted as the task of computing a most preferred diagnosis Δ for the diagnosis problem $\mathcal{P}_{\text{JOP}} := (\text{SD}_\kappa, \text{COMPS}, \text{OBS})$ where

- the *system description* SD_κ corresponds to the JSSP instance with deadline κ,
- the *set of system components* COMPS constitutes the original job set J of the JSSP,
- the *abnormality assumption* for a component means removing the respective job from J,

[2] Under the common assumption that the preference function p (e.g., probability) of a diagnosis is strictly antimonotone, i.e., $p(X) > p(Y)$ holds whenever $X \subset Y$ (cf. [8, Sec. 2.1.3]).

- the *set of observations* OBS is empty,
- the *consistency check* whether $\text{SD}_\kappa \cup \text{OBS} \cup \Delta$ is non-contradictory assuming $\Delta \subseteq \text{COMPS}$ to be abnormal and $\text{COMPS} \setminus \Delta$ to be normal amounts to solving a JSSP decision problem,
- the *component costs* are the given (positive) job utilities u_j for $j \in J$, and where

the preference function $p(\Delta)$ for a diagnosis Δ is defined as the sum of the utilities of jobs in $J \setminus \Delta$. Note, for uniform job utilities, minimum-cardinality diagnoses are most preferred.

This essentially means that any generally applicable best-first diagnosis computation algorithm can be applied to tackle the JOP (cf. [10] for a definition and examples of such algorithms). Based on our insights from the scheduling domain, we expect our proposed "optimization by relaxation and randomization" strategy to be particularly useful if the deterministic computation of preferred (or minimum-cardinality) diagnoses turns out to be computationally hard or impractical, and/or if highly parallel implementations are possible.

References

1 Philippe Baptiste. On minimizing the weighted number of late jobs in unit execution time open-shops. *EJOR*, 149(2):344–354, 2003. doi:10.1016/S0377-2217(02)00759-2.

2 Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, Malgorzata Sterna, and Jan Weglarz. *Handbook on Scheduling: From Theory to Practice*. Springer, 2019.

3 Giacomo Da Col and Erich C. Teppan. Industrial size job shop scheduling tackled by present day CP solvers. In *Principles and Practice of Constraint Programming (CP)*, 2019.

4 Michael Garey, David Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117–129, 1976. doi:10.1287/MOOR.1.2.117.

5 Joao Marques-Silva, Mikoláš Janota, and Anton Belov. Minimal sets over monotone predicates in boolean formulae. In *Int'l Conf. on Computer Aided Verification (CAV)*, 2013.

6 Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

7 Patrick Rodler. A formal proof and simple explanation of the QuickXplain algorithm. *Artif. Intell. Rev.*, 55(8):6185–6206, 2022. doi:10.1007/S10462-022-10149-W.

8 Patrick Rodler. Memory-limited model-based diagnosis. *Artif. Intell.*, 305:103681, 2022. doi:10.1016/J.ARTINT.2022.103681.

9 Patrick Rodler. Random vs. Best-First: Impact of Sampling Strategies on Decision Making in Model-Based Diagnosis. In *AAAI Conf. on Artificial Intelligence (AAAI)*, 2022.

10 Patrick Rodler. How should I compute my candidates? A taxonomy and classification of diagnosis computation algorithms. In *European Conf. on Artificial Intelligence (ECAI)*, 2023.

11 Patrick Rodler and Erich Teppan. The scheduling job-set optimization problem: A model-based diagnosis approach. In *Int'l Workshop on Principles of Diagnosis (DX)*, 2020. arXiv:2009.11142. arXiv:2009.11142.

12 Patrick Rodler, Erich Teppan, and Dietmar Jannach. Randomized problem-relaxation solving for over-constrained schedules. In *Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR)*, 2021. doi:10.24963/kr.2021/72.

13 Linus Schrage. A more portable Fortran random number generator. *TOMS*, 5(2):132–138, 1979. doi:10.1145/355826.355828.

14 Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation. In *European Conf. on Artificial Intelligence (ECAI)*, 2014.

15 Eric Taillard. Benchmarks for basic scheduling problems. *EJOR*, 64(2):278–285, 1993.

Summary of "Sequence-Oriented Diagnosis of Discrete-Event Systems"

Gianfranco Lamperti
Department of Information Engineering, University of Brescia, Italy

Stefano Trerotola
Department of Information Engineering, University of Brescia, Italy

Marina Zanella
Department of Information Engineering, University of Brescia, Italy

Xiangfu Zhao
School of Computer and Control Engineering, Yantai University, China

Abstract

What follows is an extended abstract of a paper recently published in the *Journal of Artificial Intelligence Research* [1]. Broadly speaking, the approaches to model-based diagnosis of discrete-event systems (DESs) can be accommodated within a spectrum ranging from no to total knowledge-compilation. That paper presents three approaches, one for either end of the spectrum, and one falling in between, in order to compute diagnoses that are sequences of faults.

In the literature, model-based diagnosis has always been conceived as *set-oriented*, meaning that a candidate is a *set* of faults, or faulty components, that explains a collection of observations. This perspective applies to both static and dynamical systems. Diagnosis of DESs is no exception: a candidate is traditionally a set of faults occurring in a trajectory of the DES that conforms with a given sequence of observations, namely a *temporal observation*. To improve diagnostic explanation and support decision making, a *sequence-oriented* perspective to diagnosis of DESs is presented, where a candidate is a *sequence* of faults occurring in a trajectory of the DES that produces the temporal observation. That candidate, called a *fault sequence*, differs from a classical candidate mainly in three ways: (*i*) it includes the *multiset* of faults occurred in the trajectory, where all the occurrences of the same fault are encompassed, (*ii*) it provides a total temporal order of faults, and (*iii*) its length may be unbounded, as the same fault may occur an unlimited number of times in the trajectory. Therefore, in a sequence-oriented perspective, the diagnosis output is the (possibly infinite) set of all distinct fault sequences, each relevant to a (possibly infinite) set of trajectories of the DES that produce a given temporal observation. Having a possibly unbounded set of candidates contrasts with set-oriented diagnosis, where the set of candidates is bounded by the powerset of the domain of faults. Still, a possibly unbounded set of fault sequences is shown to be a regular language, which can be defined as a regular expression over the domain of faults: this regular expression represents the output of sequence-oriented diagnosis.

The additional information on the chronological order of faults and their multiple occurrences embedded in a fault sequence may be important for ranking purposes, as well as for supporting diagnosticians and/or troubleshooting algorithms to make decisions in critical scenarios and, possibly, to take some sequential diagnosis steps. Since the output of sequence-oriented diagnosis includes additional pieces of information with respect to the output of set-oriented diagnosis, two questions raise quite naturally: *how can the sequence-oriented task be performed, and which is its performance?* The paper addresses the former question for a class of DESs usually considered by the authors (*active systems*) in three ways, and shows some experimental evidence to answer the latter.

Specifically, three sound and complete techniques to perform the task of *monitoring-based* diagnosis are described and compared, where a new candidate set is generated at the occurrence of each observation, namely: (1) *blind* diagnosis, with *no* compiled knowledge, (2) *greedy* diagnosis, with *total* knowledge compilation, and (3) *lazy* diagnosis, with *partial* knowledge compilation. By *knowledge* we mean a data structure, slightly similar to a classical DES diagnoser, which is called an *explainer*, that can be generated (compiled) either entirely offline (greedy diagnosis) or incrementally online (lazy diagnosis). Among them, only the blind diagnosis engine relies solely on the model of the DES, without producing any data structure offline nor performing any offline processing.


© Gianfranco Lamperti, Stefano Trerotola, Marina Zanella, and Xiangfu Zhao;
licensed under Creative Commons License CC-BY 4.0
35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa; Article No. 34; pp. 34:1–34:2

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

It constructs and iteratively updates the portion of a DES behavioral space that conforms with the temporal observation perceived so far. To compute the resulting regular expression, a state-elimination algorithm has been exploited from the literature, which takes as input a non-deterministic automaton and generates the regular expression of the accepted language. In greedy diagnosis, the explainer generated offline has a size that is smaller than the size of the classical diagnoser that enables the online computation of set-oriented candidates. However, its smaller size comes with a disadvantage: the explainer is a non-deterministic finite automaton, whereas the diagnoser is deterministic. Hence, in greedy diagnosis, whenever a new observable event is perceived, the time required by the online search within the explainer may be longer than the search time in the diagnoser. Moreover, building a complete explainer is out of the question for real size DESs, owing to the exponential explosion of the states involved. The building blocks of the explainer are called *fault spaces*. Each fault space is generated by a variant of the state-elimination algorithm in the literature, whose pseudocode is specified in the paper. In the construction of a complete explainer, the generation of the regular expressions is very expensive since there is a trade-off between the richness of the information embedded in the diagnosis output and the time required for its computation. The paper includes some hints about the asymptotic time complexity of the algorithms. A possible mitigation of the complexity comes from lazy diagnosis, which performs only partial knowledge compilation, thus obtaining a *partial explainer*. The three diagnosis engines have been implemented, with the software code being open source (see [1] for the link). The experimental activity recorded in the paper indicates that only lazy diagnosis may be viable in non-trivial application domains, both for the construction of the partial explainer and for the online computation of the candidates. A partial explainer can be progressively extended, the upgrade being relevant either just to a single observable event (the latest one perceived during monitoring, in case it is not encompassed yet by the current partial explainer) or to the observation pattern corresponding to a behavioral scenario. Typically, behavioral scenarios can be defined to represent frequent and/or critical evolutions of the DES that we want to detect quickly during monitoring via compiled knowledge.

A sample application inspired by a small real-world example in the literature is described in the paper. The aim is to show that in the considered domain of Labour Market, as well as in other similar domains, sequence-oriented diagnosis is actually more convenient than set-oriented diagnosis.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics

Keywords and phrases model-based reasoning, monitoring-based diagnosis, discrete-event systems, sequence-oriented diagnosis, active systems, knowledge-compilation, laziness

Digital Object Identifier 10.4230/OASIcs.DX.2024.34

Category Extended Abstract

Funding This work was supported in part by the National Natural Science Foundation of China (grant number 61972360).

References

1 G. Lamperti, S. Trerotola, M. Zanella, and X. Zhao. Sequence-oriented diagnosis of discrete-event systems. *Journal of Artificial Intelligence Research*, 78:69–141, 2023. doi:10.1613/jair.1.14630.