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.

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.

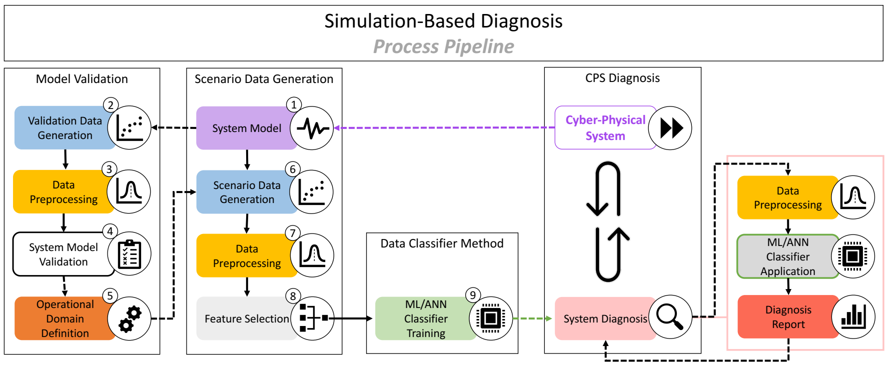


Figure 1 Simulation-based Diagnosis general pipeline process.

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

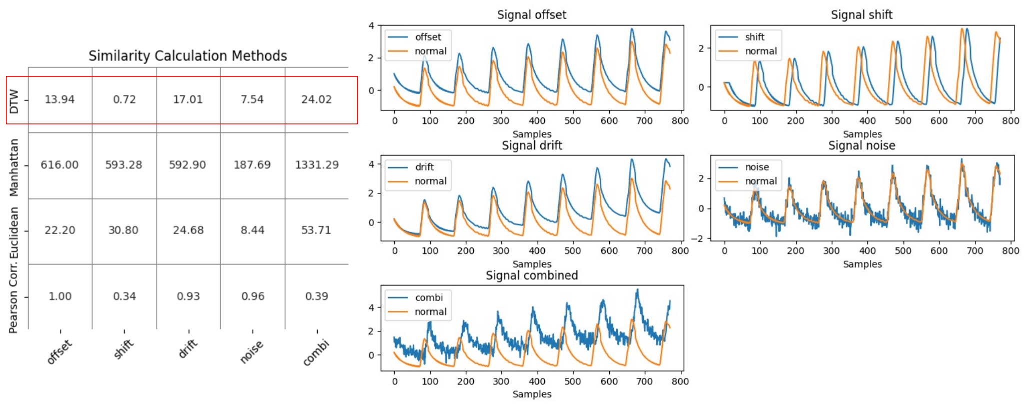
[24] for data processing and ML or ANN algorithms. We make use of the FMU simulation framework [9] controlled via a Python interface based on the library called pyFMI [1]. The framework has been developed to execute simulations of CPS models in co-simulation mode. In addition, a user-friendly Python 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.



	offset	shift	drift	noise	combi
DTW	13.94	0.72	17.01	7.54	24.02
Manhattan	616.00	593.28	592.90	187.69	1331.29
Euclidean	22.20	30.80	24.68	8.44	53.71
Pearson Corr.	1.00	0.34	0.93	0.96	0.39

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.

Distinguishabliliity 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.

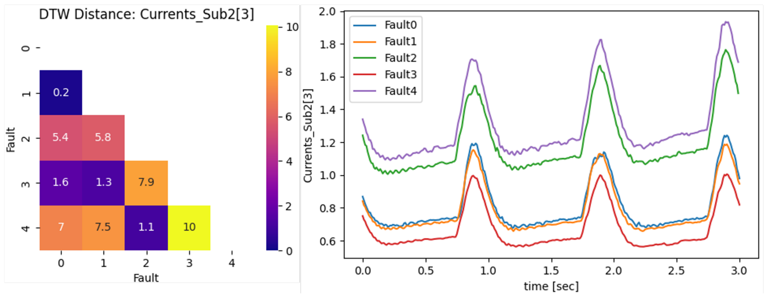


Figure 3 DTW matrix comparison between different fault configurations in the measured data for $Currents\_Sub2[3]$. A value close to 0 means highly matching, and increasing values mean a deviation in the signals. The DTW matrix shows the signal for a load of 20 NM, a speed between 3000 and 3650 rpm, 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

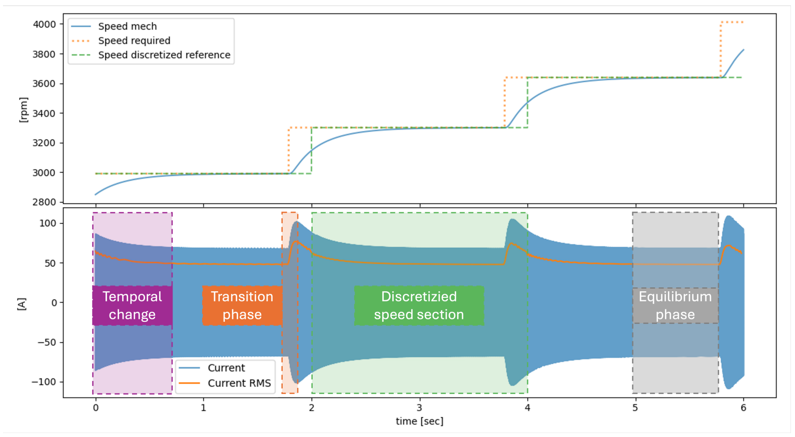


Figure 4 Identification of phases in signals and observations of a CPS.

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

```
1: function PREPROCESSING($D_{raw}$)
2:     $X, y \leftarrow D_{raw_{(\tau_{0\to k}, f_{0\to l})}}$     ▷ extract $F$ and $L$ data from raw data for all $k$ $\tau$ and all $l$ system classes $f$
3:     $\Delta w(\omega_{ref}) = \lfloor \frac{120}{\omega_{ref} * n_{poles}} * n_{periods} * n_{sampling} \rceil$     ▷ time window related to $\omega_{ref}$ to capture $n_{periods}$
   periods in waveform data (currents)
4:     $X_{stats} \leftarrow \mathbf{MTW}_{0\to n}(X_\omega, \Delta w(\omega_{ref}), \Delta s, F, metric) \quad \forall X_\omega \in X(\omega_{ref})$     ▷ $\omega_{ref}$ extracted from data to
   compute corresponding $\Delta w$ and $X_{stats}$ with $n_{stats} = \frac{n}{\Delta s}$ samples and $n_F = len(F) * len(metric)$ features
5:     $y_{stats} \leftarrow \mathbf{MTW}_{0\to n}(y_S, \Delta s, L)$     ▷ $L$ labels with $\Delta s$ with $n_{stats} = \frac{n}{\Delta s}$ samples
6:     $X_{stats,norm}, N \leftarrow$ normalization of $X_{stats}$     ▷ $N$ normalization parameters
7:     return $D \leftarrow X_{stats,norm}, y_{stats}, N$
8: end function
```

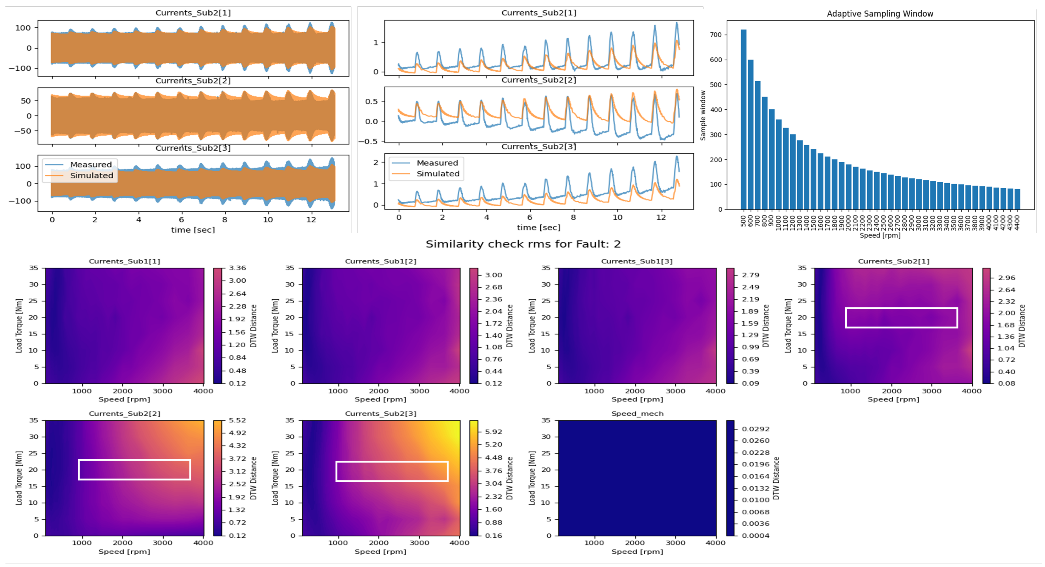


Figure 5 DTW comparison between simulated and measured data for each feature of Fault 2. Low values indicate similarity and high values indicate deviation. The DTW matrix shows three signals, each for τ from 0 to 35 Nm and ω_{mech} from 1000 to 3650 rpm. The signals in the upper graphs represent τ 20 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:

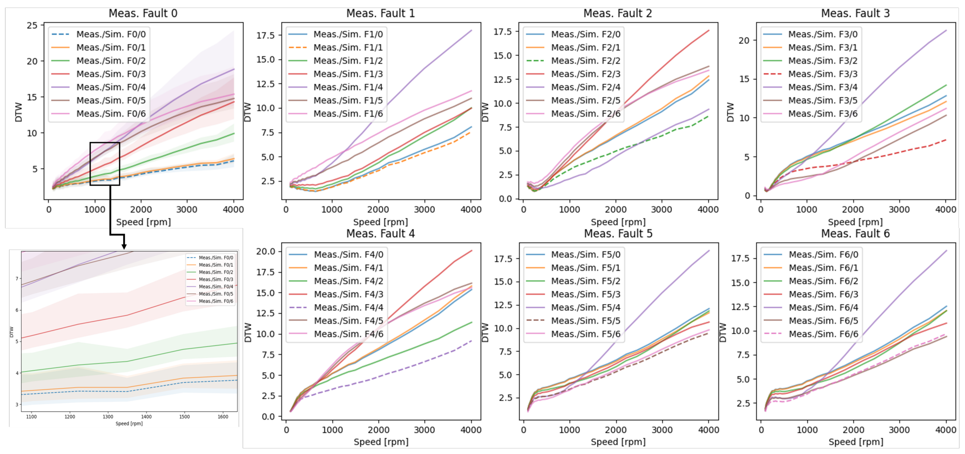


Figure 6 DTW distinguishability between measured (reference) and simulated normal and Fault 1-6 data for all features combined. The features are used on τ 0 - 35 Nm (shadowed area indicates min and max values) and ω_{mech} from 100 up to 4400 rpm. The data is normalized with Z-score.

- **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 $\mathbf{M}$

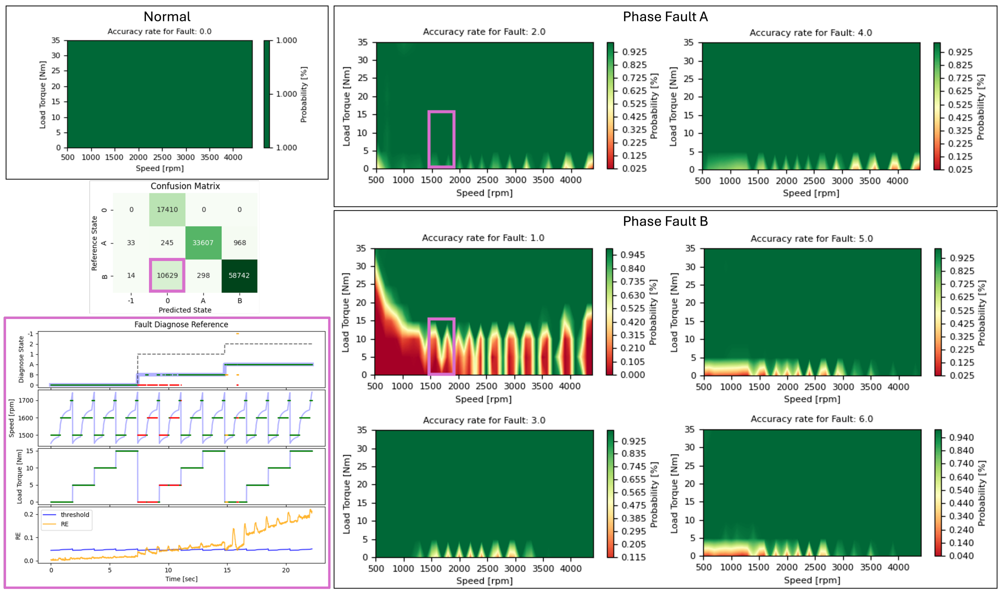
```
function Train Diagnosis Methods($D_S, D_M$)
    $X_{S_0} \leftarrow D_{S_{(\tau_{0\rightarrow k}, f_0)}}$                ▷ extract normal simulated data for all $\tau$
    $X_{S_f}, y_{S_f} \leftarrow D_{S_{(\tau_{0\rightarrow k}, f_{1\rightarrow l})}}$    ▷ extract fault injected simulated data for all $\tau$
    $X_{M_0} \leftarrow D_{M_{(\tau_{0\rightarrow k}, f_0)}}$                ▷ extract normal measured data for all $\tau$
    $\mathbf{A} \leftarrow$ 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 $X_\omega$ is a matrix with $n$ samples and $m$ features, $\omega$ 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 $\omega$ and factor $r$
    with: $\mu_{\varepsilon_\omega} = MEAN(\varepsilon_\omega)$ , $\sigma_{\varepsilon_\omega} = STD(\varepsilon_\omega)$
    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 $\Theta_\omega$ based on $\omega$
    $\mathbf{M} \leftarrow$ 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($D_{M_{raw}}$, $metric$, $\Delta w$, $\Delta s$, $N_S$)
    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
        $X_{\Delta_i} \leftarrow X_{M_{i-\Delta w\rightarrow i}}$
        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 $X_{\Delta_i}$
            $\overline{\omega} \leftarrow mean(X_{\Delta_{i,\omega_{mech}}})$    ▷ extract the window data of feature $\omega_{mech}$ 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].



	-1	0	A	B
0	0	17410	0	0
A	33	245	33607	968
B	14	10629	298	58742

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`.