# An Interval Constraint Programming Approach for Quasi Capture Tube Validation

**Abderahmane Bedouhene**
LIGM, Ecole des Ponts ParisTech, Université Gustave Eiffel, CNRS, Marne-la-Vallée, France

**Bertrand Neveu**
LIGM, Ecole des Ponts ParisTech, Université Gustave Eiffel, CNRS, Marne-la-Vallée, France

**Gilles Trombettoni**
LIRMM, Université de Montpellier, CNRS, France

**Luc Jaulin**
Lab-STICC, ENSTA-Bretagne, Brest, France

**Stéphane Le Menec**
MBDA, Le Plessis Robinson, France

## Abstract

Proving that the state of a controlled nonlinear system always stays inside a time moving bubble (or capture tube) amounts to proving the inconsistency of a set of nonlinear inequalities in the time-state space. In practice however, even with a good intuition, it is difficult for a human to find such a capture tube except for simple examples. In 2014, Jaulin et al. established properties that support a new interval approach for validating a quasi capture tube, i.e. a candidate tube (with a simple form) from which the mobile system can escape, but into which it enters again before a given time. A quasi capture tube is easy to find in practice for a controlled system. Merging the trajectories originated from the candidate tube yields the smallest capture tube enclosing it.

This paper proposes an interval constraint programming solver dedicated to the quasi capture tube validation. The problem is viewed as a differential CSP where the functional variables correspond to the state variables of the system and the constraints define system trajectories that escape from the candidate tube "for ever". The solver performs a branch and contract procedure for computing the trajectories that escape from the candidate tube. If no solution is found, the quasi capture tube is validated and, as a side effect, a corrected smallest capture tube enclosing the quasi one is computed. The approach is experimentally validated on several examples having 2 to 5 degrees of freedom.

**2012 ACM Subject Classification** Applied computing → Operations research; Mathematics of computing → Ordinary differential equations; Mathematics of computing → Differential algebraic equations; Mathematics of computing → Interval arithmetic; Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint satisfaction problem, Interval analysis, Dynamical systems, Contractor

**Digital Object Identifier** 10.4230/LIPIcs.CP.2021.18

**Funding** This work was supported by the French Agence Nationale de la Recherche (ANR) [grant number ANR-16-CE33-0024].

**Acknowledgements** We also thank our colleagues, Alexandre Goldsztejn and Alessandro Colotti, for the exchange of ideas and their kind help on the experiments.


27th International Conference on Principles and Practice of Constraint Programming (CP 2021).
Editor: Laurent D. Michel; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# 1 Introduction

Many mobile robots such as wheeled robots, boats, or planes are described by differential equations. For this type of robots, it is difficult to prove some properties such as the avoidance of collisions with some moving obstacles. This is even more difficult when the initial condition is not known exactly or when some uncertainties occur.

A Graal would be to compute a *capture tube* (or equivalently a *positive invariant* tube [24]), i.e. a time moving "bubble" (a set-valued function associating to each time $t$ a subset of $\mathbb{R}^n$) from which a feasible trajectory cannot escape. The definitions and properties of capture tubes have been studied by several authors [2, 4], but the algorithms for their computation are almost absent except in the linear case [33, 25, 11]. In the nonlinear case, approaches based on interval analysis [19, 31] or Lipschitz assumptions [32] have also been investigated, but the performances are poor if no propagation techniques are used. When time is discrete, efficient algorithms are given in [35], but they cannot be extended to robotics systems described by differential equations.

Instead, a satisfactory alternative is to present a candidate tube to a tool that could validate whether it is a capture tube or not. This validation problem can generally be transformed into proving the inconsistency of a constraint system by combining guaranteed integration and Lyapunov theory [26, 36]. Unfortunately, when the system dynamics is complex, even with a good intuition, it is difficult for a human to present a significant capture tube because of its irregular form.

Jaulin et al. proposed in [13] an original approach based on interval analysis. The idea is to validate a *quasi* capture tube, also called *periodic invariant set* [17], i.e. a candidate tube (with a simple form) from which the mobile system can escape, but into which it can enter again before a given time. Merging these trajectories with the candidate tube computes the smallest capture tube enclosing the quasi capture one. Jaulin et al. established properties that support this new approach, but the algorithms were not described and were validated only on a simple pendulum example with two degrees of freedom. Their approach worked in two steps, where the first one focused on the crossout constraints (see Section 3) while the second step managed the other constraints.

The contribution presented in this paper is built upon those properties. Compared to Jaulin et al. approach, the solver follows a pure CSP approach expressing the quasi capture tube validation problem, where the domains are tubes defined recently in the Tubex-Codac library [28, 30]. After the background in Section 2, we formally define in Section 3 the quasi capture tube validation problem and its expression as a CSP. We then propose a Branch and Contract solver dedicated to this problem in Section 4 and show in Section 5 how it scales up on several problems from 2 to 5 state dimensions.

# 2 Background

We first provide some background about intervals, inclusion functions and contraction. We then briefly present how intervals can be used to handle dynamical systems.

## 2.1 Intervals

Contrary to numerical analysis methods that work with single values, interval methods can manage sets of values enclosed in intervals. Interval methods are known to be particularly useful for handling nonlinear constraint systems.

▶ **Definition 1** (Interval, box, box size/diameter). *An interval* $[x_i] = [\underline{x_i}, \overline{x_i}]$ *defines the set of reals* $x_i$ *such that* $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ *denotes the set of all intervals. A box* $[\boldsymbol{x}]$ *denotes a Cartesian product of intervals* $[\boldsymbol{x}] = [x_1] \times ... \times [x_n]$. *The size, width or diameter of a box* $[\boldsymbol{x}]$ *is given by* $Diam([\boldsymbol{x}]) \equiv \max_i(Diam([x_i]))$ *where* $Diam([x_i]) \equiv \overline{x_i} - \underline{x_i}$. *The midpoint* $mid([x_i])$ *of* $[x_i]$ *is* $\frac{\underline{x_i}+\overline{x_i}}{2}$.

*Interval arithmetic* [22] has been defined to extend to $\mathbb{IR}$ the usual mathematical operators over $\mathbb{R}$. For instance, the interval sum is defined by $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$. When a function $\boldsymbol{f}$ is a composition of elementary functions, an *inclusion function* $[\boldsymbol{f}]$ of $\boldsymbol{f}$ must be defined to ensure a conservative image computation. There are several inclusion functions. The *natural* inclusion function of a real function $f$ corresponds to the mapping of $f$ to intervals using interval arithmetic. For instance, the natural inclusion function $[f]_N$ of $f(x) = x(x+1)$ in the domain $[x] = [0, 1]$ computes $[f]_N([0, 1]) = [0, 1] \cdot [1, 2] = [0, 2]$. Another inclusion function is based on an interval Taylor form [12].

Interval arithmetics can be used for solving the *numerical CSP* (NCSP), *i.e.* finding solutions to an NCSP network $P = (\boldsymbol{x}, [\boldsymbol{x}], \boldsymbol{c})$, where $\boldsymbol{x}$ is an $n$-set of variables taking their real values in the domain $[\boldsymbol{x}]$ and $\boldsymbol{c}$ is an $m$-set of numerical constraints using operators like $+$, $-$, $\times$, $a^b$, exp, log, sin, *etc.* NCSP solvers, like Gloptlab [10] or Ibex [6] to name a few, follow a Branch and Contract method to solve an NCSP. The branching operation subdivides the search space by recursively bisecting variable intervals into two subintervals and exploring both sub-boxes independently. The combinatorial nature of this tree search is not always observed thanks to the *contraction* (filtering) operations applied at each node of the search tree. Informally, a contraction applied to an NCSP instance can reduce the variables domains without losing any solution.

A contractor used in this paper is the well-known `HC4-revise` [3, 21], also called *forward-backward*. This contractor handles a single numerical constraint and obtains a (generally non optimal [7]) contracted box including all the solutions of that constraint.

To contract a box w.r.t. an NCSP instance, the HC4 algorithm performs a (generalized) AC3-like propagation loop applying iteratively the HC4-Revise procedure on each constraint individually until a quasi fixpoint is obtained in terms of contraction.

CID-consistency [34] is a stronger consistency enforced on an NCSP. The CID algorithm calls its `VarCID` procedure on all the NCSP variables for enforcing the CID-consistency. `VarCID` splits a variable interval in $k$ subintervals, and runs a contractor, such as HC4, on the corresponding sub-boxes. The smallest box including the $k$ sub-boxes contracted is finally returned. The 3BCID contractor used in this paper uses a variant of the `VarCID` procedure.

## 2.2 Dynamical CSP and tubes

Intervals can also be used to handle dynamical systems that handle functional variables, also called *trajectories*.

A trajectory, denoted $\boldsymbol{x}(\cdot) = (x_1(\cdot), .., x_n(\cdot))$, is a function from $[t_0, t_f] \subset \mathbb{R}$ to $\mathbb{R}^n$. The input (argument) of $\boldsymbol{x}(\cdot)$ is named *time* in this article (and denoted $\cdot$ or $t$) while the output (image) is called *state*.

Interval methods can compute trajectories as solutions of a *differential* CSP instance.

▶ **Definition 2** (Differential CSP). *A differential CSP network is defined by* $(\boldsymbol{x}(\cdot), [\boldsymbol{x}](\cdot), \boldsymbol{c})$, *where* $\boldsymbol{x}(\cdot)$ *is a trajectory variable of domain* $[\boldsymbol{x}](\cdot)$ *and* $\boldsymbol{c}$ *denotes the set of differential constraints between variables* $\boldsymbol{x}(\cdot)$.

*Solving a differential CSP instance consists in finding the set of trajectories in* $[\boldsymbol{x}](\cdot)$ *satisfying* $\boldsymbol{c}$.

Domains of a differential CSP network are *tubes*, set-valued functions associating to each time $t$ a subset of $\mathbb{R}^n$, on which we apply contraction and bisection operations.

▶ **Definition 3** (Tube [15]). *A tube* $[\boldsymbol{x}](\cdot) : [t_0, t_f] \to \mathcal{P}(\mathbb{R}^n)$ *is an interval of two trajectories* $[\underline{\boldsymbol{x}}(\cdot), \overline{\boldsymbol{x}}(\cdot)]$ *such that* $\forall t \in [t_0, t_f]$, $\underline{\boldsymbol{x}}(t) \leqslant \overline{\boldsymbol{x}}(t)$. *We also consider empty tubes that depict an absence of solutions.*
*A trajectory* $\boldsymbol{x}(\cdot)$ *belongs to the tube* $[\boldsymbol{x}](\cdot)$ *if* $\forall t \in [t_0, t_f]$, $\boldsymbol{x}(t) \in [\boldsymbol{x}](t)$.

Fig. 1 illustrates a one-dimensional tube ($[t_0, t_f] \to \mathcal{P}(\mathbb{R})$) enclosing a trajectory $x(\cdot)$.

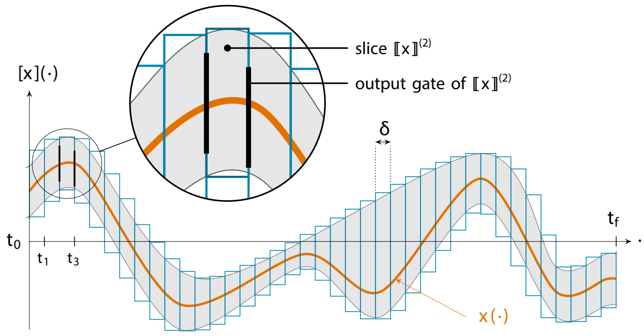


**Figure 1** A one-dimensional tube $[x](\cdot)$ Courtesy of S. Rohou). In grey enclosing a random trajectory $x(\cdot)$ depicted in plain line (orange). $[x](\cdot)$ is an interval of two functions $[\underline{x}(\cdot), \overline{x}(\cdot)]$. The tube is numerically represented by a set of $\delta$-width slices illustrated by blue boxes.

A tube is represented numerically by a set of boxes corresponding to temporal slices. More precisely, an $n$-dimensional tube $[\boldsymbol{x}](\cdot)$ with a sampling time $\delta > 0$ is implemented as a box-valued function which is constant for all $t$ inside intervals $[k\delta, k\delta + \delta]$, $k \in \mathbb{N}$. The box $[k\delta, k\delta + \delta] \times [\boldsymbol{x}](t_k)$, with $t_k \in [k\delta, k\delta + \delta]$, is called the $k^{th}$ *slice* of the tube $[\boldsymbol{x}](\cdot)$ and is denoted by $[\![\boldsymbol{x}]\!]^{(k)}$. This implementation takes rigorously into account floating-point precision when building a tube: computations involving $[\boldsymbol{x}](\cdot)$ will be based on its slices, thus giving a reliable outer approximation of the solution set. The slices may be of same width as depicted in Fig. 1, but the tube can also be implemented with a customized temporal *slicing*. Finally, we endow the definition of a slice $[\![\boldsymbol{x}]\!]^{(k)}$ with the *slice (box) envelope* (blue painted in Fig. 1) and two input/output *gates* $[\boldsymbol{x}](t_k)$ and $[\boldsymbol{x}](t_{k+1})$ (black painted) that are intervals of $\mathbb{IR}^n$ through which trajectories are entering/leaving the slice.

Once a tube is defined, it can be handled in the same way as an interval. We can for instance use arithmetic operations as well as function evaluations. If $f$ is an elementary function such as sin, cos or exp, we define $f([x](\cdot))$ as the smallest tube containing all feasible values: $f([x](\cdot)) = \big[\{f(x(\cdot)) \mid x(\cdot) \in [x](\cdot)\}\big]$.

The Branch & Contract algorithm presented in this paper makes choice points on tubes [28], defined as follows and illustrated by Fig. 2.

▶ **Definition 4** (Tube bisection). *Let* $[\boldsymbol{x}](\cdot)$ *be a tube of a trajectory* $\boldsymbol{x}(\cdot)$ *defined over* $[t_0, t_f]$. *Let* $t_k$ *be an instant in* $[t_0, t_f]$, $i$ *a dimension in* $\{1..n\}$, *and* $[x_i]$ *the interval value of* $[x_i](\cdot)$ *at* $t_k$. *Let* $mid(x_i)$ *be* $\frac{\overline{x_i} + \underline{x_i}}{2}$.
*The tube bisection* $(t_k, i)$ *of* $[\boldsymbol{x}](\cdot)$ *produces two tubes* $[\boldsymbol{x}^L](\cdot)$ *and* $[\boldsymbol{x}^R](\cdot)$ *equal to* $[\boldsymbol{x}](\cdot)$ *except at time* $t_k$, *where* $[x_i^L] = [\underline{x_i}, mid(x_i)]$ *and* $[x_i^R] = [mid(x_i), \overline{x_i}]$.

In practice, a bisection $(t_k, i)$ is applied only to a gate of the tube. For the particular problem handled in this paper, $t_k$ will always be $t_0$.

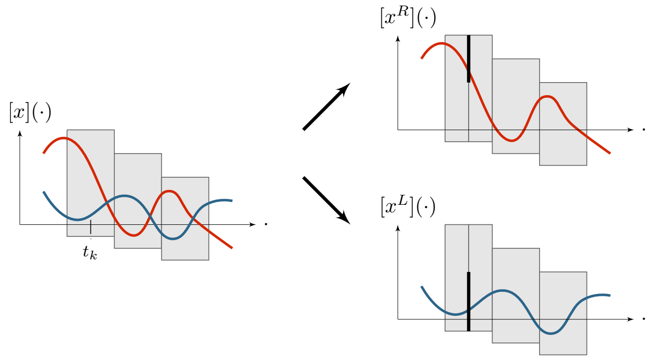


**Figure 2** Illustration of a tube bisection at time $t_k$ (courtesy of S. Rohou). A gate is created at $t_k$ and the two sub-tubes $[\boldsymbol{x}^L](\cdot)$ and $[\boldsymbol{x}^R](\cdot)$ differ only by their new created sub-gate (in bold). Two (among an infinity) possible trajectories of the initial tube are separated by the bisection, one belonging to $[\boldsymbol{x}^L](\cdot)$, the other belonging to $[\boldsymbol{x}^R](\cdot)$.

There exist several types of differential constraints. The problem presented in Section 3 contains only well-known ordinary differential equations (ODEs).

▶ **Definition 5** (Ordinary differential equation – ODE). *Consider* $\boldsymbol{x}(\cdot) : [t_0, t_f] \to \mathbb{R}^n$*, its derivative* $\dot{\boldsymbol{x}}(\cdot) : [t_0, t_f] \to \mathbb{R}^n$*, and an* evolution function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$*, possibly non-linear. An ODE is defined by:* $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$

This means that for all times $t$ in the temporal domain $[t_0, t_f]$ the derivative of the function $\boldsymbol{x}$ depends only on the state $\boldsymbol{x}$ at time $t$ and on the time $t$. An ODE can be used to define a well-known IVP differential system or an extension.

▶ **Definition 6** (IVP, interval IVP). *The initial value problem (IVP) is defined by an ODE* $\dot{\boldsymbol{x}}(\cdot) = \boldsymbol{f}\big(\boldsymbol{x}(\cdot)\big)$ *and an initial condition* $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$*, where* $\boldsymbol{x}_0$ *is a constant in* $\mathbb{R}^n$*.*
*In an interval IVP, the initial condition is bounded by a box, i.e.* $\boldsymbol{x}(t_0) \in [\boldsymbol{x}_0]$*.*

The IVP is studied for hundreds years and can be solved by numerous numerical methods, e.g. the Euler method [5]. The interval IVP can be solved by interval analysis tools, such as VNODE [23], CAPD [14], COSY [27] and `DynIbex` [8]. These solvers are also called *Guaranteed Integration (GI)* solvers. GI solvers use different algorithms to rigorously integrate the initial information over time. In particular, the CAPD tool used in our solver combines a high-order interval Taylor form to integrate the state from an instant to a next one, and a step limiting the wrapping effect implied by interval calculation: it encloses the solution at gates by an envelope sharper than a box, such as rotated boxes [20].

## 3 Quasi Tube Capture Validation as a CSP

In automatic control, validation of *stability* properties of dynamical systems is an important and difficult problem [16]. A tube $\mathbb{G}(t)$ is *positive invariant* (or a *capture* tube) for a dynamic system $\boldsymbol{x}(.)$ if all the possible trajectories of $\boldsymbol{x}(.)$ remain in $\mathbb{G}(t)$ for ever, i.e. for every time in the temporal domain defined.

▶ **Definition 7** (Capture tube[1]). *Let $S_{\boldsymbol{f}}$ be a dynamic system defined by an ODE $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$. Let $\mathbb{G}(t)$ be a tube defined by an inequality $\{\boldsymbol{x}(t) \mid g(\boldsymbol{x}(t), t) \leq 0\}$, where $g : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is a differentiable function w.r.t. $\boldsymbol{x}$ and $t$.*

*Then:*

*$\mathbb{G}(t)$ is said to be a capture tube for $S_{\boldsymbol{f}}$ if:* $\quad \boldsymbol{x}(t_i) \in \mathbb{G}(t_i), \tau > 0 \implies \boldsymbol{x}(t_i + \tau) \in \mathbb{G}(t_i + \tau)$

Conditions can be checked to validate whether a given tube is a capture tube or not.

▶ **Theorem 1** (Cross-out conditions [13]). *Let $S_{\boldsymbol{f}}$ be a dynamic system defined by $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$, and a tube $\mathbb{G}(t) = \{\boldsymbol{x}(t) \mid g(\boldsymbol{x}(t), t) \leq 0\}$. Consider the constraint system:*

$$\begin{cases} (i) & \frac{\partial g(\boldsymbol{x},t)}{\partial \boldsymbol{x}} . \boldsymbol{f}(\boldsymbol{x}, t) + \frac{\partial g(\boldsymbol{x},t)}{\partial t} \geq 0 \\ (ii) & g(\boldsymbol{x}, t) = 0 \end{cases} \tag{1}$$

*If (1) is inconsistent (i.e., $\forall \boldsymbol{x}, \forall t \geq 0$, (1) has no solution), then $\mathbb{G}(t)$ is a capture tube.*

The constraint system (1) describes the subset of $S_{\boldsymbol{f}}$ trajectories that escape from $\mathbb{G}(t)$. If this subset is empty, it means that $\mathbb{G}(t)$ is a capture tube.

In [13], Jaulin et al. highlighted that it is not easy for the user to define "by hand" a relevant capture tube of irregular form and propose rather to ask for a so-called *quasi* capture tube of simple form. Informally, some trajectories can escape from a quasi capture tube, but can enter into it again later, i.e. before a given horizon $t_f$. Such a trajectory satisfies the following constraints:

- $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t) \quad$ ($\boldsymbol{x}(t)$ is a trajectory of $S$)
- $\exists t_0 \in [t_0]$, $\boldsymbol{x}(t_0)$ satisfies (1) $\quad$ ($\boldsymbol{x}(t)$ exits from $\mathbb{G}(t)$ at $t_0 \in [t_0]$)
- $\exists t_{in} \in \, ]t_0, t_f]$ s.t. $x(t_{in}) \in \mathbb{G}(t_{in}) \quad$ ($\boldsymbol{x}(t)$ goes back inside $\mathbb{G}(t)$ at $t_{in}$)

Instead of using these constraints directly, the idea of this paper is to propose a CSP expressing the "negation" of the quasi capture problem, and to detail a Branch & Contract method to solve it.

▶ **Definition 8** (CSP defining the quasi capture validation problem). *Let $S_{\boldsymbol{f}}$ be a dynamic system defined by $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), t\big)$, and a candidate tube $\mathbb{G}(t) = \{\boldsymbol{x}(t) \mid \boldsymbol{g}(\boldsymbol{x}(t), t) \leq 0\}$.*

*The constraint network $N = (\boldsymbol{x}(.), [\boldsymbol{x}(.)], \boldsymbol{c})$ defines the quasi capture validation problem, where $\boldsymbol{x}(.)$ describes the system living in the domain/tube $[\boldsymbol{x}(.)]$, and $\boldsymbol{c}$ includes the three following (vectorial) constraints:*

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t) & \textit{(differential constraint)} \\ \exists t_0,\ \boldsymbol{x}(t_0) \textit{ satisfies } (1) & \textit{(cross out constraint)} \\ \forall t \in \, ]t_0, t_f] \ g(\boldsymbol{x}(t), t) > 0 & \textit{(escape constraint)} \end{cases}$$

The constraints model the fact that the system can escape from $\mathbb{G}(t)$ "for ever", i.e. cannot go back in $\mathbb{G}(t)$ before $t_f$. If $N$ is inconsistent, then it proves that $\mathbb{G}(t)$ is a quasi capture tube.

Furthermore, consider the trajectories that satisfy the cross out constraint but violate the escape constraint. It is straightforward to check that if the CSP has no solution, adding these trajectories to the candidate (quasi capture) tube builds a capture tube [13].

---

[1] For the sake of clarity, and because our application problems fall in this case, we restrict ourselves to the case where $\mathbb{G}(t)$ is defined by only one inequality. The corresponding cross out constraint system is slightly more complicated otherwise [13], but the solver presented in the next section also works on it.

## 4 Branch and Contract Algorithm

In this section, we describe a branch and contract algorithm for solving the differential CSP defined above. More precisely, Algorithm 1 computes a set $OutList$ of tubes including all the system trajectories that escape from the candidate tube $\mathbb{G}(t)$ "for ever", i.e. at a time greater than $t_0$ and remaining outside $\mathbb{G}(t)$ until $t_f$.

### 4.1 Main algorithm

The initial domain $initTube$ is $[t_0, t_f] \times [x]$, where $[x]$ is a big or infinite box initializing the state variables. The other input parameters are the candidate capture tube $\mathbb{G}(t)$, a precision parameter on the time ($timestep$) and vectorial parameters $\boldsymbol{\epsilon}_{start}$ and $\boldsymbol{\epsilon}_{min}$, that specify the diameters of all variables at the initial gate. They are detailed further.

**Algorithm 1** Branch and contract.

1 Input $(\mathbb{G}(t), initTube, t_0, t_f, timestep, \boldsymbol{\epsilon}_{start}, \boldsymbol{\epsilon}_{min})$
2 Output ($OutList$ : list of solution tubes ; $UndeterminedList$ : list of "small" tubes still undetermined)
3 $tubes \leftarrow \{initTube\}$
4 **while** $(tubes \neq \emptyset)$ **do**
5     $tube \leftarrow$ `Pop`$(tubes)$
6     (ContractionResult, $tube$) $\leftarrow$ `Contraction`($tube$, S, $\mathbb{G}(t)$, $t_0$, $t_f$, $timestep$, $\boldsymbol{\epsilon}_{start}$)
7     **if** $(ContractionResult =$ `out`$)$ **then**
8         $OutList \leftarrow OutList \cup \{tube\}$
9     **else if** $(ContractionResult = undetermined)$ **then**
10         **if** $Diam(tube(t_0)) \leq \boldsymbol{\epsilon}_{min}$ **then**
11             $UndeterminedList \leftarrow UndeterminedList \cup \{tube\}$
12         **else**
13             $(tube_{left}, tube_{right}) \leftarrow$ `Bisect`($tube$, bisectionStrategy)
14             $tubes \leftarrow \{tube_{left}\} \cup \{tube_{right}\} \cup tubes$
15         **end**
16     **else**
17         /* ContractionResult = `in`: Nothing to do : $tube$ is discarded because its trajectories all enter inside $\mathbb{G}(t)$ at an instant in $[t_0, t_f]$ */
18     **end**
19 **end**

Algorithm 1 follows a tree search that combinatorially subdivides the initial domain $initTube$ into smaller tubes, in depth-first order. At each node of the search tree handling a $tube$, a contraction is achieved using the three types of constraints detailed above. The function `Contraction` (Line 6) returns a contracted tube and a status ContractionResult associated to it. $tube$ can become empty (and ContractionResult = `in`) if `Contraction` could prove that the tube in entirely inside $\mathbb{G}(t)$ at an instant between $t_0$ and $t_f$ (see Lines 16–18). A second case occurs when $tube$ has been detected outside $\mathbb{G}(t)$ after a time and until $t_f$ (Line 7). It is not useful to subdivide $tube$ further because all the trajectories inside $tube$ are solutions. Therefore $tube$ is stored in $OutList$. The last case corresponds to an internal node of the search tree and occurs when the contraction cannot decide one of the cases "in" or "out" above (Line 9). If $tube$ is sufficiently large (Line 12), the branching operation bisects *tube* in two sub-tubes $tube_{left}$ and $tube_{right}$ and pushed them in front of *tubes* (depth first order). The tube bisection is performed at the first gate (at $t_0$) because one has the most information at this time (cross out conditions hold). Note it is sufficient to perform all the bisections at the same time because with an ODE an "instanciation" at one time allows one to deduce the trajectory perfectly.

No more bisection is achieved if the *tube* size has reached a given precision $\epsilon_{min}$, and *tube* is stored in a list of "undetermined" tubes (Line 11). Algorithm 1 stops when *tubes* is empty. If *OutList* and *UndeterminedList* are empty, then $\mathbb{G}(t)$ is a quasi invariant tube for the system $S$.

We detail in Algorithm 2 the different contractors applied to the current *tube*. *tube* is first contracted by the cross out constraints (Line 3). `CrossoutContraction` contracts *tube* at time $t_0$ according to the cross out constraints. It calls the state-of-the-art contractors HC4 [3] and 3BCID [18, 34] on the cross out constraint subsystem (see Section 5 describing the experiments).

With the call to `ODEEvalContraction` (Line 6), we then proceed with the contraction of the differential (ODE) constraint and the escape constraint. Note that this contraction procedure is run only under a given level of the search tree, where, for each dimension, the tube diameter at $t_0$ is lower than the user parameter $\epsilon_{start}$. Indeed, this differential contraction during the time window $[t_0, t_f]$ is costly and needs a relatively small input box (initial condition) to efficiently contract *tube*, with the help of guaranteed integration.

**Algorithm 2** Function `Contraction` called by Algorithm 1.

1 **Function** `Contraction`($S$, $\mathbb{G}(t)$, *tube*, $t_0$, $t_f$, *timestep*, $\epsilon_{start}$)
2     *tube* $\leftarrow$ `CrossOutContraction`(*tube*, $S$, $\mathbb{G}(t)$)
3     **if** *(tube* $= \emptyset$*)* **then**
4         ContractionResult $\leftarrow$ `in`
5     **else if** $(Diam(tube(t_0)) < \epsilon_{start})$ **then**
6         ContractionResult $\leftarrow$ `ODEEvalContraction`($S$, *tube*, $\mathbb{G}(t)$, $t_0$, $t_f$, *timestep*)
7     **else**
8         ContractionResult $\leftarrow$ `undetermined`
9     **end**
10     **return** (ContractionResult, *tube*)
11 **end**

## 4.2 Differential contraction

White box differential contractors, e.g. the `ctcDeriv` and `ctcEval` contractors available in the Tubex/Codac free library [29], could be used to contract *tube* w.r.t. the ODE and escape constraints.

Instead, for performance reasons, we preferred to exploit a state-of-the-art guaranteed integration (GI) tool, like VNODE-LP [23] or CAPD [14], to benefit from its optimized internal representations. The corresponding method is described in Algorithm 3.

The `ODEEvalContraction` function contracts *tube* by integrating the ODE from $t_0$ to $t_f$ using the CAPD GI solver. The function `GI_Simulation` (Line 5) calls the GI solver with the interval initial value $tube(t_i)$, the *tube* gate at time $t_i$. The GI generally needs to construct several gates before reaching $t_f$, and `GI_Simulation` allows one to incrementally build the next *slice* between $t_i$ and a computed time $t_{i+1}$. By doing this integration, the

**Algorithm 3** Function `ODEEvalContraction` called by Algorithm 2.

```
Function ODEEvalContraction(S, tube, 𝔾(t), t0, tf, timestep)
    ti ← t0
    tout ← ∞
    repeat
        (slice, ti+1) ← GI_Simulation(S, tube(ti), ti, tf)
        (ContractionResult, tout) ← GI_Eval (slice, 𝔾(t), timestep, ti, ti+1, tout)
        tube[ti, ti+1] ← tube[ti, ti+1] ∩ slice
        ti ← ti+1
    until (ti = tf) or (ContractionResult = in)
    if ContractionResult = in or tout ≠ ∞ then
        return ContractionResult
    else
        return undetermined
    end
end
```

GI solver builds an associated high-order Taylor polynomial that can be evaluated rapidly at any gate or subslice inside $[t_i, t_{i+1}]$. This is the task achieved by `GI_Eval`. Without detailing, `GI_Eval` splits $[t_i, t_{i+1}]$ into contiguous subslices of (time) size *timestep* and tests whether *tube* during the studied subslice satisfies the escape (from $\mathbb{G}(t)$) constraint or not. In the latter case, the integration is interrupted (Algorithm 3 stops) and ContractionResult is set to `in`. The whole *tube* is rejected. If a subslice satisfies the escape constraint, $t_{out}$ is used to memorize the first instant where it occurs. If $t_{out} = \infty$, then $t_{out}$ is set to $t_i$. If a subsequent subslice evaluation does not return `out`, then $t_{out}$ is set back to $\infty$. Indeed, recall that a solution tube must satisfy the escape constraint in all times from $t_{out}$ to $t_f$. When $t_f$ is reached, only two cases are still possible. Either *tube* has escaped from $\mathbb{G}(t)$ at $t_{out}$ until $t_f$ (a solution), or *tube* has intersected $\mathbb{G}(t)$ at some instants, including $t_f$. In that case, we cannot conclude and the result of the contraction will be `undetermined`. Figure (3) summarizes the different cases described above.

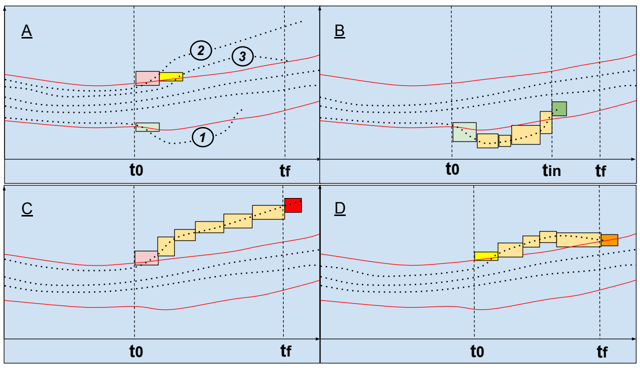


**Figure 3** Different tubes built by the solver. Three particular trajectories (1), (2) and (3) are highlighted in the figure. **A**: First slice satisfying the cross out constraints corresponding to the three trajectories leaving the tube $\mathbb{G}(t)$. **B**: A tube enclosing (1) is integrated and is getting inside $\mathbb{G}(t)$. **C**: A tube enclosing (2) is escaping from $\mathbb{G}(t)$. **D**: An undetermined tube enclosing (3): the algorithm cannot conclude.

Another possible case not described in the pseudo-code is when `GI_Simulation` fails to compute a part of the simulation. This result is equivalent to the `undetermined` result since the algorithm is not able to conclude if the *tube* goes inside $\mathbb{G}(t)$ or not. The choice of $\boldsymbol{\epsilon}_{start}$ has a significant impact on the frequency of this "pathological" case (see experiments).

## 4.3 Discussion

Algorithm 1 provides two main answers. The favorable case is when the solver returns no solution: `OutList` and `Underdeterminedlist` are empty. The algorithm is correct and guarantees that $\mathbb{G}(t)$ is a quasi capture tube. Furthermore, as a side effect, by merging with $\mathbb{G}(t)$ all the `in` tubes rejected by the algorithm, we can build the smallest (i.e., inclusion-wise minimal) capture tube including the quasi capture tube. The second case occurs when the solver computes a non empty `OutList` or `Underdeterminedlist`. This corresponds generally to the computation of a non quasi capture tube but, theoretically, it is possible that a trajectory could enter inside $\mathbb{G}(t)$ after $t_f$, before $t_{out}$ (`OutList` $\neq$ *emptyset*), or during the undetermined temporal slices. In this sense, the solver is not complete while these numerical issues occur rarely provided $t_f$ is large enough (according to the command of the system), and the precision size $\epsilon_{min}$ is sufficiently small.

# 5 Experiments

The current section presents some results provided by an implementation of Algorithm 1 that significantly improves a first code called `Bubbibex` and written to validate the pendulum problem [1]. This new `Bubbibex` is implemented in C++. It uses the IBEX library [6] with the HC4 [3] and 3BCID [18, 34] contractors for propagating the cross out conditions constraints. It also uses the CAPD/DynSys library for the differential contractor based on guaranteed integration [14] and the Tubex/CODAC library for tube structure [29].

Experiments have been carried out using an Intel(R) Xeon(R) CPU E3-1225 V2 at 3.20GHz. In each experiment, see Table [1], we highlight the results obtained by the solver when we tried different values for one so-called observed parameter ($\boldsymbol{\epsilon}_{start}$ or bubble radius or etc.).

These responses include the running time of each experiment (CPU-Time) in second, and the number of computed tubes corresponding to the leaves of the search tree: "In" for tubes getting inside $\mathbb{G}(t)$,"Und" for undetermined tubes and "Out" for tubes staying out of $\mathbb{G}(t)$ at $t_f$. These numbers are reported in the tables presenting the results of each experiment.

The simulation time of each experiment is at most $t_f = 100$ with $timestep = 0.01$. The bisection strategy used is the `Maximum Diam Ratio`, selecting the variable $[x_i]$ with the greatest ratio $Diam([x_i])/\epsilon_i$.

▶ Remark 2. Rewriting a non autonomous ODE as an autonomous ODE adds the temporal variable "t" to the state variables, increasing the dimension of the problem by 1. As a result, the dimension of the vectorial parameters $\boldsymbol{\epsilon}_{start}$ and $\boldsymbol{\epsilon}_{min}$ might increase if the domain of the temporal variable "t" defined by $[t_0]$ is not a degenerate interval (i.e. $[\underline{t_0} < \overline{t_0}]$).

## 5.1 Pendulum

$$P : \begin{cases} \dot{x} = y \\ \dot{y} = -sin(x) - \rho.y \end{cases} \tag{2}$$

**Table 1** Characteristics of the different experiments.

| Problem | Type | State variables | Bubble |
|---|---|---|---|
| Pendulum | Non Linear | 2 | Static |
| 2D Linear system | Linear | 2 | Dynamic |
| Tracking | Linear | 2 and 3 | Static and Dynamic |
| Pursuit game | Non Linear | 3 and 5 | Dynamic |

Let $P$ be a dynamical system describing the motion of a pendulum, where $x$ is the angular position, $y$ is the angular velocity and $\rho = 0.15$ the constant friction coefficient of the pendulum. We want to find a quasi capture tube for the system $P$.

**Table 2** Parameters of the pendulum experiment.

| First gate | Bubble | $r_0$ | Observed parameter |
|---|---|---|---|
| $x, y \in [-10, 10]$ | $x^2 + y^2 - r_0^2 \leq 0$ | 1 | $\boldsymbol{\epsilon}_{start}$ |

When $\boldsymbol{\epsilon}_{start}$={1,1} (Line 1 of Table 3), the differential contractor is not able to successfully contract the tube. This is due to a large initial condition that prevents the guaranteed integration from computing a solution and leads the solver to bisect the initial gate of the tube before reaching the right precision. Having a good intuition on the parameter $\boldsymbol{\epsilon}_{start}$ (Line 2 of Table 3) can improve the efficiency of the method. The CSP has no solution, the studied bubble is a quasi capture tube.

**Table 3** Results for pendulum system.

| $\boldsymbol{\epsilon}_{start}$ | $\boldsymbol{\epsilon}_{min}$ | In | Und | Out | CPU |
|---|---|---|---|---|---|
| {1,1} | {0.1,0.1} | 6 | 0 | 0 | 72.2 |
| {0.5,0.5} | {0.1, 0.1} | 6 | 0 | 0 | 0.00734 |

## 5.2 2D linear system

$$R : \begin{cases} \dot{x} = u_1 \\ \dot{y} = u_2 \end{cases} \tag{3}$$

Let $R$ be a robot described by the linear dynamical system (3) such that $(x, y)$ is the position and $u_1 = -x + t$, $u_2 = -y$ the controllers.

We want the robot to stay inside a dynamic bubble.

For bubbles with radius $r_0 \geq 1.2$, the solver is able to verify that they are capture tubes (the cross-out constraint contracts to an empty domain).

Table 5 depicts the results obtained with bubbles having a constant radius $r_0 = 1.1$, $r_0 = 1$ or $r_0 = 0.9$ or a time dependent radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$. For instance, for $[t_0] = 0$, we can prove that, for $r_0 = 0.9$, the bubble is not a quasi capture tube, but we are not able to conclude for $r_0 = 1$, even for a small $\boldsymbol{\epsilon}_{min}$. It is therefore not necessary for $r_0 = 1$ and for $r_0 = 0.9$ to perform the experiment for $[t_0] = [0, 100]$ since the bubble cannot be proved to be a quasi capture tube. On the other hand, the bubble with a radius $r_0 = 1.1$, and the bubble with an increasing radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$ are quasi capture tubes for all $t_0$ in $[0, 100]$.

**Table 4** Parameters of the 2D linear system experiment.

| First gate | Bubble | Observed parameter |
|---|---|---|
| $x, y \in [-100, 100]$ | $(x-t)^2 + (y)^2 - r_0^2 \leq 0$ | $r_0$ |

**Table 5** Results for $r_0 = 1.1$, $r_0 = 1$, $r_0 = 0.9$ and $r_0 = \frac{1}{\sqrt{5}}(1+t)$.

| $r_0$ | $[t_0]$ | $\epsilon_{start}$ | $\epsilon_{min}$ | In | Und | Out | CPU |
|---|---|---|---|---|---|---|---|
| 1.1 | [0, 100] | {1,1,0.1} | {0.1,0.1,0.01} | 2048 | 0 | 0 | 2.6 |
| 1 | 0 | {1,1} | {0.1,0.1} | 0 | 2 | 0 | 0.08 |
| 1 | 0 | {1,1} | {1e-8,1e-8} | 0 | 10 | 0 | 0.91 |
| 0.9 | 0 | {1,1} | {0.1,0.1} | 0 | 0 | 2 | 0.05 |
| $\frac{(1+t)}{\sqrt{5}}$ | [0, 100] | {1,1,0.1} | {0.1,0.1,0.01} | 7 | 0 | 0 | 0.04 |

## 5.3 Linear tracking system

Consider the following linear dynamical system:

$$\dot{\boldsymbol{x}}(t) = A(\boldsymbol{x}(t) - \boldsymbol{T}(t)) \tag{4}$$

with $\boldsymbol{x}(t)$ the tracking system and $\boldsymbol{T}(t)$ the target.

We want to study the stability of the system (4) by finding a quasi capture tube. We will study two cases for the system (4), one with a static bubble centered at the origin, and the other one with a dynamic bubble centered at the target.

## 2D and 3D tracking systems

Consider the 2D linear system:

$$n = 2: \quad A = \begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos(t) \\ \sin(2t) \end{bmatrix} \tag{5}$$

and the 3D linear system:

$$n = 3, \quad A = \begin{bmatrix} 1 & 3 & 0 \\ -3 & -2 & -1 \\ 0 & 1 & -3 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos(t) \\ \cos(t)\sin(2t) \\ -\sin(t)\sin(2t) \end{bmatrix} \tag{6}$$

**Table 6** Parameters of the linear tracking system experiment.

| First gate | Bubble | $r_0$ | Observed parameter |
|---|---|---|---|
| $x_1, x_2 \in [-10, 10]$ | $x_1^2 + x_2^2 - r_0^2 \leq 0$ | 2 | Dim/Bubble |
| $x_1, x_2 \in [-10, 10]$ | $(x_1 - T_1(t))^2 + (x_2 - T_2(t))^2 - r_0^2 \leq 0$ | 2 | Dim/Bubble |
| $x_1, x_2, x_3 \in [-10, 10]$ | $x_1^2 + x_2^2 + x_3^2 - r_0^2 \leq 0$ | 2 | Dim/Bubble |
| $x_1, x_2, x_3 \in [-10, 10]$ | $(x_1 - T_1(t))^2 + (x_2 - T_2(t))^2 + (x_3 - T_3(t))^2 - r_0^2 \leq 0$ | 2 | Dim/Bubble |

Both targets, in the 2D and 3D linear systems, have a periodic pattern movement and their period is $2\pi$. We can then restrict the study of the stability of both systems to $t_0 \in [0, 2\pi]$ by setting the time domain of the initial gate to $[t_0] = [0, 2\pi]$.
From Table 7 we can conclude that both bubbles are quasi capture tubes for the system (4). Fig. 4 illustrates the 3D tracking system.

**Table 7** Results for both systems (2D and 3D) and both bubbles (static and dynamic).

| Dim | Bubble | $\epsilon_{start}$ | $\epsilon_{min}$ | In | Und | Out | CPU |
|---|---|---|---|---|---|---|---|
| 2D | Static | {1,1,0.05} | {0.1,0.1,0.01} | 370 | 0 | 0 | 1.20 |
| 2D | Dynamic | {1,1,0.05} | {0.1,0.1,0.01} | 1021 | 0 | 0 | 1.65 |
| 3D | Static | {1,1,1,0.05} | {0.1,0.1,0.1,0.01} | 3290 | 0 | 0 | 7.10 |
| 3D | Dynamic | {1,1,1,0.05} | {0.1,0.1,0.1,0.01} | 4040 | 0 | 0 | 11.94 |

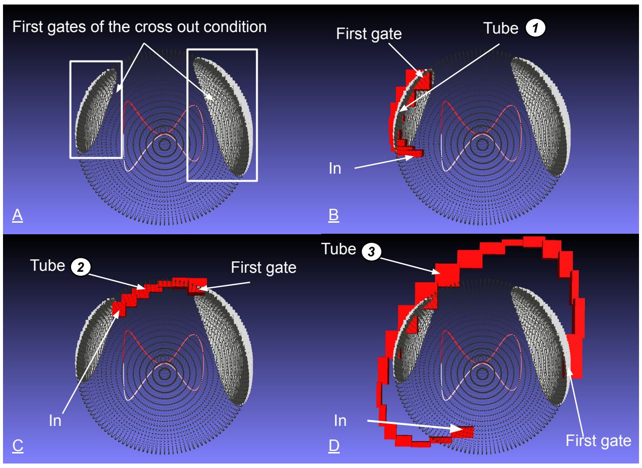


**Figure 4** Sample of tubes of the 3D linear tracking system leaving the static bubble. The figure illustrates the bubble and the tubes in the state dimensions. **A**: First gates satisfying the cross out constraints appear in white on the spherical bubble of radius $r_0 = 2$. The periodic target, with an "$\infty$" trajectory, appears in the center of the bubble. Its color is going from red at $t = 0$ to white at $t = 2\pi$. **B** and **C**: Tubes (in red) getting almost immediately inside the sphere. **D**: Tube going far away from the sphere and finally landing after one first unsuccessful landing trial.

## 5.4 Pursuit evasion game

A "pursuit evasion" game is a situation where a pursuer ($P$) wants to catch an evader ($E$) trying to escape from him. In the following experiment, we will present two problems based the "pursuit evasion" game, one in the plane, and the other one in the 3D-space. The evader ($E$) will be at the center of a dynamic bubble, and we want the pursuer to stay inside the bubble in order to catch the evader. In other words, we want the bubble to be a capture tube, or at least, a quasi capture tube.

### Pursuit game on the plane

Consider the following pursuer $P$ and evader $E$:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta) \\ \dot{y} = u_1.sin(\theta) \\ \dot{\theta} = u_2 \end{cases} \quad E : \begin{cases} x_d = v.t \\ y_d = sin(\rho t) \end{cases} \tag{7}$$

where $x$ and $y$ are the position and $\theta$ the heading of $P$.

The velocity of the pursuer and its heading are respectively controlled by $u_1 = ||n||$ and $u_2 = -K.sin(\theta - \theta_d)$ such that $\theta_d = atan2(n)$ and $n$ is defined as follows:

$$n = \begin{bmatrix} n_x \\ n_y \end{bmatrix} = \frac{1}{dt} \begin{bmatrix} x_d - x \\ y_d - y \end{bmatrix} + \begin{bmatrix} \dot{x}_d \\ \dot{y}_d \end{bmatrix}$$

**Table 8** Parameters of the pursuit game on the plane experiment.

| First gate | Bubble | $r_0$ | Observed parameter |
|---|---|---|---|
| $x, y \in [-10, 10]$, $\theta \in [0, 2\pi]$ | $(x - x_d)^2 + (y - y_d)^2 - r_0^2 = 0$ | 1 | $\epsilon_h$ |

We add the following constraint on the heading of the pursuer:

$$h(x, y, \theta, t) = (cos(\theta) - cos(\theta_d))^2 + (sin(\theta) - sin(\theta_d))^2 - \epsilon_h \leq 0$$

Constants: $K = 1$, $v = 7$, $\rho = 1$, $dt = 1$

## Pursuit Evasion game in the 3D-space

Let the pursuer $P$ and the evader $E$:

$$P : \begin{cases} \dot{x} = u_1.cos(\theta).cos(\psi) \\ \dot{y} = u_1.cos(\theta).sin(\psi) \\ \dot{z} = u_1.sin(\theta) \\ \dot{\psi} = u_2 \\ \dot{\theta} = u_3 \end{cases} \qquad E : \begin{cases} x_d = v.w.t \\ y_d = v.w.sin(w.t) \\ z_d = -v.w.cos(w.t) \end{cases} \tag{8}$$

where $x$, $y$ and $z$ are the position, $\psi$ is the circular rotation speed and $\theta$ is the vertical rotation speed of P. The controls $u_1 = ||n||$, $u_2 = K(\psi - \psi_d)$ and $u_3 = K(\theta - \theta_d)$. Without going into details, $\theta_d$ and $\psi_d$ are defined with analytical expressions.

$$n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \frac{1}{dt} \begin{bmatrix} x_d - x \\ y_d - y \\ z_d - z \end{bmatrix} + \begin{bmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{z}_d \end{bmatrix}$$

We have added the following constraints on the circular and vertical rotations of the pursuer:

$$h_1(\psi, t) = (cos(\psi) - cos(\psi_d))^2 + (sin(\psi) - sin(\psi_d))^2 - \epsilon_h \leq 0$$

$$h_2(\theta, t) = (cos(\theta) - cos(\theta_d))^2 + (sin(\theta) - sin(\theta_d))^2 - \epsilon_h \leq 0$$

Constants: $v = 2$, $w = 1$, $K = 10$, $dt = 1$.

**Table 9** Parameters of the pursuit game in the 3D-space experiment.

| First gate | Tube candidate | $r_0$ | Observed parameter |
|---|---|---|---|
| $x, y, z \in [-10, 10]$, $\theta, \psi \in [0, 2\pi]$ | $(x - x_d)^2 + (y - y_d)^2 + (z - z_d)^2 - r_0^2 = 0$ | 1 | $\epsilon_h$ |

## Pursuit evasion game results

Here again, both evaders follow a periodic pattern of period $2\pi$, so the study is restricted to a time domain $t_0 \in [0, 2\pi]$.

When the problem scales up (in the number of the state variables, number of nonlinearities, system stiffness, etc.), the solver faces some difficulties. We can see in Tables 10 and 11 how varies the number of tubes computed by the solver for validating a quasi capture tube (the reader can compare with the previous experiments). The number of tubes required can be drastically lowered by using small values for parameter $\epsilon_h$ that restrict the initial heading (resp. circular and vertical rotations) of the pursuer (see Fig. 5).

**Table 10** Results of the pursuit game on the plane show that, with a small parameter $\epsilon_h$, we can validate the quasi capture tube on the whole period of the evader.

| $[t_0]$ | $\epsilon_h$ | $\epsilon_{start}$ | $\epsilon_{min}$ | In | Und | Out | CPU |
|---|---|---|---|---|---|---|---|
| 0 | 0.02 | $\{0.1, 0.1, 0.1\}$ | $\{0.01, 0.01, 0.005\}$ | 129 | 0 | 0 | 1.74 |
| $[0, 2\pi]$ | 0.02 | $\{0.1, 0.1, 0.1, 0.05\}$ | $\{0.01, 0.01, 0.005, 0.005\}$ | 16672 | 0 | 0 | 585 |
| 0 | 0.2 | $\{0.1,0.1,0.1\}$ | $\{0.01, 0.01, 0.005\}$ | 437 | 0 | 0 | 8.01 |
| $[0, 2\pi]$ | 0.2 | $\{0.1, 0.1, 0.1, 0.05\}$ | $\{0.01, 0.01, 0.005, 0.005\}$ | 105735 | 0 | 0 | 6561 |

**Table 11** Results of the pursuit game in 3D-space: even for small parameter value $\epsilon_h$, studying one tenth of the period for $[t_0]$ requires a huge CPU time. On the other hand, the quasi capture tube is then validated.

| $[t_0]$ | $\epsilon_h$ | $\epsilon_{start}$ | $\epsilon_{min}$ | In | Und | Out | CPU |
|---|---|---|---|---|---|---|---|
| 0 | 0.1 | $\{0.1,0.1,0.1,0.05,0.05\}$ | $\{0.01,0.01,0.01,0.005,0.005\}$ | 21414 | 0 | 0 | 590 |
| 0 | 0.08 | $\{0.1, 0.1, 0.1, 0.05, 0.05\}$ | $\{0.01, 0.01, 0.01, 0.005, 0.005\}$ | 8128 | 0 | 0 | 236 |
| 0 | 0.0625 | $\{0.1, 0.1, 0.1, 0.05, 0.05\}$ | $\{0.01, 0.01, 0.01, 0.005, 0.005\}$ | 1852 | 0 | 0 | 62.4 |
| 0 | 0.05 | $\{0.1, 0.1, 0.1, 0.05, 0.05\}$ | $\{0.01, 0.01, 0.01, 0.005, 0.005\}$ | 176 | 0 | 0 | 11.1 |
| $[0, \frac{\pi}{5}]$ | 0.05 | $\{0.1, 0.1, 0.1, 0.05, 0.05, 0.05\}$ | $\{0.01, 0.01, 0.01, 0.005, 0.005, 0.005\}$ | 241103 | 0 | 0 | 109 |

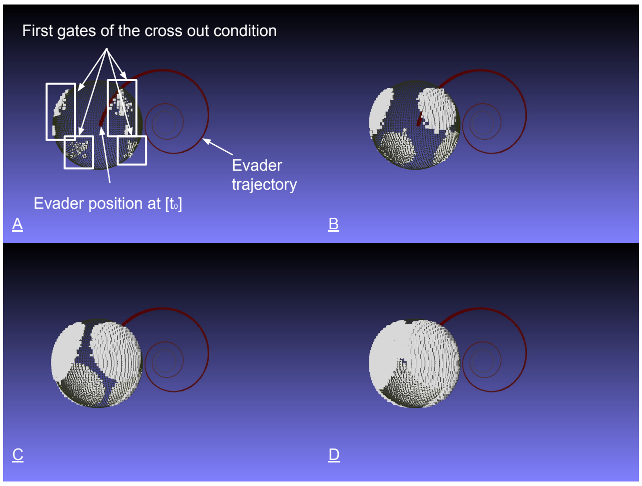


**Figure 5** Pursuit evasion game in 3D-space: Illustration of the bubble and the evader trajectory (in red) in the state dimensions. First gates satisfying the cross out constraints at $[t_0] = 0$ appear in white on the spherical bubble of radius $r_0 = 1$, centered on the position of the evader at $[t_0] = 0$. We can notice how the number of gates varies for different values for $\epsilon_h$. **A**: $\epsilon_h = 0.05$. **B**: $\epsilon_h = 0.0625$. **C**: $\epsilon_h = 0.08$. **D**: $\epsilon_h = 0.1$

## 6 Conclusion

We have proposed a Branch and Contract solver dedicated to the quasi capture tube validation, a problem for which the algorithms are almost absent. The solver is sufficiently generic to handle different problems. The performance of the solver is based on filtering/contraction algorithms and on the use of the guaranteed integration solver CAPD for the integration of differential equations. We have validated the solver in different application examples scaling from 2 to 5 state dimensions. To simplify the problem, the solver can accept additional constraints on the command parameters. We have tried to propagate domain reductions backward (from the escape constraint deductions to $t_0$) with no success. Nevertheless, there is still improvement space for future work. We could improve the shape of the capture tube candidate using Lyapunov approaches or parametric barrier functions [9]. We could also improve our algorithm for computing in an auto-adaptive manner the $\epsilon_{start}$ parameter deciding the tube size under which it is relevant to run the guaranteed integration solver. Finally, we could improve our software using a multi threading approach.

## References


1 A. Akkouche, J.-B. Bénéfice, Q. Bréfort, B. Desrochers F. Carbonera, T. Issautier, M. Laranjeira-Moreira, V. Le Doze, D. Monnet, and A. Oubelhaj. BUBBIBEX with IBEX. Engineering internship report, ENSTA Bretagne, 2014. URL: `https://www.ensta-bretagne.fr/jaulin/archirob.html#bm_2013_14`.

2 J.-P. Aubin. Viability Kernels and Capture Basins of Sets Under Differential Inclusions. *SIAM Journal on Control and Optimization*, 40(3):853–881, 2001. `doi:10.1137/S036301290036968X`.

3 F. Benhamou, F. Goualard, L. Granvilliers, and J.F. Puget. Revising Hull and Box Consistency. In D. De Schreye, editor, *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, pages 230–244. MIT Press, 1999.

4 F. Blanchini and S. Miani. *Set Theoretic Methods in Control.* Birkhauser, 2008.

5 J.C. Butcher. *Numerical Methods for Ordinary Differential Equations.* Wiley, 2004. URL: `https://books.google.fr/books?id=okzpIwEX8aEC`.

6 G. Chabert. IBEX – an Interval-Based EXplorer, 2020. URL: `http://www.ibex-lib.org/`.

7 H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3):213–228, 1999.

8 Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge–Kutta Methods. *Reliable Computing*, 22(1):79–103, July 2016.

9 A. Djaballah, A. Chapoutot, M. Kieffer, and O. Bouissou. Construction of Parametric Barrier Functions for Dynamical Systems using Interval Analysis. *Automatica*, 78:287–296, 2017. `doi:10.1016/j.automatica.2016.12.013`.

10 F. Domes. GLOPTLAB: a Configurable Framework for the Rigorous Global Solution of Quadratic Constraint Satisfaction Problems. *Optimization Methods & Software*, 24:727–747, October 2009. `doi:10.1080/10556780902917701`.

11 A. Girard, C. Le Guernic, and O. Maler. Efficient Computation of Reachable Sets of Linear Time-invariant Systems with Inputs. *Hybrid Systems: Computation and Control*, 3927:257–271, 2006.

12 E. R. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, New York, NY, 1992.

13 L. Jaulin, D. Lopez, V. Le Doze, S. Le Menec, J. Ninin, G. Chabert, M. S. Ibnseddik, and A. Stancu. Computing Capture Tubes. In Marco Nehmeier, Jürgen Wolff von Gudenberg, and Warwick Tucker, editors, *Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 209–224, Cham, 2016. Springer International Publishing.

14 T. Kapela, M. Mrozek, D. Wilczak, and P. Zgliczynski. CAPD: : Dynsys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems. *CoRR*, abs/2010.07097, 2020. `arXiv:2010.07097`.

15 F. Le Bars, J. Sliwka, L. Jaulin, and O. Reynet. Set-membership State Estimation with Fleeting Data. *Automatica*, 48(2):381–387, 2012. `doi:10.1016/j.automatica.2011.11.004`.

16 S. Le Menec. Linear Differential Game with Two Pursuers and One Evader. *Advances in Dynamic Games*, 11:209—226, 2011.

17 Y. I. Lee and B. Kouvaritakis. Constrained Robust Model Predictive Control Based on Periodic Invariance. *Automatica*, 42:2175–2181, 2006.

18 O. Lhomme. Consistency Techniques for Numeric CSPs. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 232–238. Morgan Kaufmann, 1993. URL: `http://ijcai.org/Proceedings/93-1/Papers/033.pdf`.

19 M. Lhommeau, L Jaulin, and L. Hardouin. Inner and Outer Approximation of Capture Basins using Interval Analysis. *ICINCO 2007*, 2007.

20 R. Lohner. Enclosing the Solutions of Ordinary Initial and Boundary Value Problems. In E. Kaucher, U. Kulisch, and Ch. Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. BG Teubner, Stuttgart, Germany, 1987.

21 F. Messine. *Méthodes d'Optimisation Globale basées sur l'Analyse d'Intervalle pour la Résolution des Problèmes avec Contraintes.* PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.

22 R. E. Moore. *Interval Analysis*, volume 4. Prentice-Hall Englewood Cliffs, 1966.

23 N. S. Nedialkov, K. R. Jackson, and J. D. Pryce. An Effective High-Order Interval Method for Validating Existence and Uniqueness of the Solution of an IVP for an ODE. *Reliable Computing*, 7(6):449–465, 2001. doi:10.1023/A:1014798618404.

24 S. Olaru, J.A. De Dona, M.M. Seron, and F. Stoican. Positive Invariant Sets for Fault Tolerant Multisensor Control Schemes. *International Journal of Control*, 83(12):2622–2640, 2010.

25 S. V. Rakovic, E. C. Kerrigan, K. I. Kouramas, and D. Q. Mayne. Invariant approximations of the minimal robust positively invariant set. *IEEE Trans. Autom. Control*, 50(3):406–410, 2005.

26 S. Ratschan and Z. She. Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-like Functions. *SIAM J. Control and Optimization*, 48(7):4377––4394, 2010.

27 N. Revol, K. Makino, and M. Berz. Taylor Models and Floating-point Arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64:135–154, 2005.

28 S. Rohou, A. Bedouhene, G. Chabert, A. Goldsztejn, L. Jaulin, B. Neveu, V. Reyes, and G. Trombettoni. Towards a Generic Interval Solver for Differential-Algebraic CSP. In *Proc. CP, Constraint Programming, Springer, LNCS 12333*, pages 864–879. Springer, 2020.

29 S. Rohou et al. The Tubex Library – Constraint-programming for robotics, 2021. URL: http://simon-rohou.fr/research/tubex-lib/.

30 Simon Rohou, Luc Jaulin, Lyudmila Mihaylova, Fabrice Le Bars, and Sandor M. Veres. Guaranteed Computation of Robot Trajectories. *Robotics and Autonomous Systems*, 93:76–84, 2017. doi:10.1016/j.robot.2017.03.020.

31 S. Romig, L. Jaulin, and A. Rauh. Using Interval Analysis to Compute the Invariant Set of a Nonlinear Closed-Loop Control System. *Algorithms*, 12(262), 2019.

32 P. Saint-Pierre. Hybrid Kernels and Capture Basins for Impulse Constrained Systems. In C.J. Tomlin and M.R. Greenstreet, editors, *in Hybrid Systems: Computation and Control*, volume 2289, pages 378–392. Springer-Verlag, 2002.

33 F. Tahir and M. Jaimoukha. Low-Complexity Polytopic Invariant Sets for Linear Systems Subject to Norm-Bounded Uncertainty. *IEEE Trans. Autom. Control*, 60:1416–1421, 2015.

34 G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, Constraint Programming, LNCS 4741*, pages 635–650. Springer, 2007.

35 J. Wan, J. Vehi, and N. Luo. A Numerical Approach to Design Control Invariant Sets for Constrained Nonlinear Discrete-time Systems with Guaranteed Optimality. *Journal of Global Optimization*, 44:395–407, 2009.

36 J. A. Yorke. Invariance for Ordinary Differential Equations. *Mathematical System Theory*, 1(4):353–372, 1967.