An Interval Constraint Programming Approach for Quasi Capture Tube Validation

Abderahmane Bedouhene
LIGM, Ecole des Ponts ParisTech, Université Gustave Eiffel, CNRS, France

Bertrand Neveu
LIGM, Ecole des Ponts ParisTech, Université Gustave Eiffel, CNRS, France

Gilles Trombettoni
LIRMM, University of Montpellier, CNRS, France

Luc Jaulin
Lab-STICC, ENSTA-Bretagne, France

Stéphane Le Menec
MBDA, France

Abstract
Proving that 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 practice however, even with a good intuition, it is difficult for a human to find a significant capture tube because of its irregular form. 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. Merging these trajectories with the candidate tube computes the minimum capture tube enclosing the candidate one.

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 minimum 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 Replace ccsdesc macro with valid one

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

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements This work was supported by the French Agence Nationale de la Recherche (ANR) [grant number ANR-16-CE33-0024]. We also thank our colleagues, Alexandre Goldsztejn and Alessandro Colotti for the exchange of ideas and their kind help the experiments.

1 Introduction

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 \([x_i] = [\bar{x}_i, \overline{x}_i]\) defines the set of reals \(x_i\) such that \(\bar{x}_i \leq x_i \leq \overline{x}_i\). \(\mathbb{R}\) denotes the set of all intervals. A box \([x]\) denotes a Cartesian product of intervals \([x] = [x_1] \times \ldots \times [x_n]\).

The size, width or diameter of a box \([x]\) is given by \(\text{Diam}([x]) = \max_i(\text{Diam}([x_i]))\) where \(\text{Diam}([x_i]) \equiv \overline{x}_i - \bar{x}_i\). The midpoint \(\text{mid}([x_i])\) of \([x_i]\) is \(\frac{\bar{x}_i + \overline{x}_i}{2}\).

Interval arithmetic [16] has been defined to extend to \(\mathbb{R}\) the usual mathematical operators over \(\mathbb{R}\). For instance, the interval sum is defined by \([x_1] + [x_2] = [\bar{x}_1 + \overline{x}_2, \bar{x}_2 + \overline{x}_1]\). When a function \(f\) is a composition of elementary functions, an inclusion function \([f]\) of \(f\) 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 \([f]\) of \(f(x) = x(x+1)\) in the domain \([x] = [0, 1]\) computes \([f]\) of \([0, 1]\) = \([0, 1] \cdot [1, 2] = [0, 2]\). Another inclusion function is based on an interval Taylor form [8].

Interval arithmetics can be used for solving the numerical CSP (NCSP), i.e. finding solutions to an NCSP network \(P = (x, [x], c)\), where \(x\) is an \(n\)-set of variables taking their real values in the domain \([x]\) and \(c\) is an \(m\)-set of numerical constraints using operators like +, −, \(\times\), \(a^b\), \(\exp\), \(\log\), \text{etc.} NCSP solvers, such as Gloptlab [7] or Ibex [4] 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 [1, 15], also called forward-backward. This contractor handles a single numerical constraint and obtains a (generally non-optimal [6]) 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 [21] is a stronger consistency enforced on an NCSP. The CID algorithm calls its \texttt{VarCID} procedure on all the NCSP variables for enforcing the CID-consistency. \texttt{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 \texttt{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 \(x(\cdot) = (x_1(\cdot), \ldots, x_n(\cdot))\), is a function from \([t_0, t_f] \subset \mathbb{R}\) to \(\mathbb{R}^n\). The input (argument) of \(x(\cdot)\) is named \textit{time} in this article (and denoted · or \(t\)) while the output (image) is called \textit{state}.

Interval methods can compute trajectories as solutions of a \textit{differential} CSP instance.
Definition 2. (Differential CSP)
A differential CSP network is defined by \( (x(\cdot), [x](\cdot), c) \), where \( x(\cdot) \) is a trajectory variable of domain \( [x](\cdot) \) and \( c \) denotes the set of differential constraints between variables \( x(\cdot) \).

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

Domains of a differential CSP network are tubes on which we apply contraction and bisection operations.

Definition 3. (Tube) [11]
A tube \([x](\cdot) : [t_0, t_f] \rightarrow IR^n\) is an interval of two trajectories \([x](\cdot), \bar{x}(\cdot)\) such that \( \forall t \in [t_0, t_f], x(t) \leq \bar{x}(t) \). We also consider empty tubes that depict an absence of solutions.

A trajectory \( x(\cdot) \) belongs to the tube \([x](\cdot)\) if \( \forall t \in [t_0, t_f], x(t) \in [x](t) \).

Fig. 1 illustrates a one-dimensional tube \([t_0, t_f] \rightarrow IR\) enclosing a trajectory \( x(\cdot) \).

A tube is represented numerically by a set of boxes corresponding to temporal slices. More precisely, an \( n \)-dimensional tube \([x](\cdot)\) 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 \([k\delta, k\delta + \delta] \times [x](t_k)\), with \( t_k \in [k\delta, k\delta + \delta] \), is called the \( k^{th}\) slice of the tube \([x](\cdot)\) and is denoted by \([x](k)\). This implementation takes rigorously into account floating-point precision when building a tube: computations involving \([x](\cdot)\) 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 \([x](k)\) with the slice (box) envelope (blue painted in Fig. 1) and two input/output gates \([x](t_k)\) and \([x](t_{k+1})\) (black painted) that are intervals of \( IR^n \) 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: \( f([x](\cdot)) = \{ f(x(\cdot)) | x(\cdot) \in [x](\cdot) \} \).

The Branch & Contract algorithm presented in this paper makes choice points on tubes [19], defined as follows and illustrated by Fig. 2.
Definition 4. (Tube bisection)
Let $x(\cdot)$ be a tube of a trajectory $\bar{x}(\cdot)$ 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 $\frac{x_i^L + x_i^R}{2}$.
The tube bisection $(t_k, i)$ of $[x](\cdot)$ produces two tubes $[x^L](\cdot)$ and $[x^R](\cdot)$ equal to $[x](\cdot)$ except
at time $t_k$, where $[x_i^L] = [x_i, \text{mid}(x_i)]$ and $[x_i^R] = [\text{mid}(x_i), x_i]$.
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 $t_0$.

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 $x(\cdot) : [t_0, t_f] \rightarrow \mathbb{R}^n$, its derivative $\dot{x}(\cdot) : [t_0, t_f] \rightarrow \mathbb{R}^n$, and an evolution function
$f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, possibly non-linear. An ODE is defined by: $\dot{x}(t) = f(x(t), t)$

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 $\dot{x}(\cdot) = f(x(\cdot))$ and an initial condition
$x(t_0) = \bar{x}_0$, where $\bar{x}_0$ is a constant in $\mathbb{R}^n$.
In an interval IVP, the initial condition is bounded by an interval, i.e. $x(t_0) \in [x_0]$. The IVP is studied for hundreds years and can be solved by numerous numerical methods,
e.g. the Euler method [3]. The interval IVP can be solved by interval analysis tools, such
as VNODE [17], CAPD [10], COSY [18] and DynIbex [5]. These solvers are also called
Guaranteed Integration (GI) solvers. GI solvers use different algorithms to rigorously simulate
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 [14].

3 Quasi Tube Capture Validation as a CSP
In automatic control, validation of stability properties of dynamical systems is an important
and difficult problem [12]. A tube $\mathcal{G}(t)$ is positive invariant (or a capture tube) for a dynamic
Definition 7. (Capture tube)
Let $S_f$ be a dynamic system defined by an ODE $\dot{x}(t) = f(x(t), t)$. Let $G(t)$ be a tube defined by an inequality $\{x(t) \mid g(x(t), t) \leq 0\}$, where $g : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^m$ is a differentiable function w.r.t. $x$ and $t$. Then:

$G(t)$ is said to be a capture tube for $S_f$ if:

\[ x(t_i) \in G(t_i), \tau > 0 \implies x(t_i + \tau) \in G(t_i + \tau) \]

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

Theorem 1. (Cross-out conditions [9])
Let $S_f$ be a dynamic system defined by $\dot{x}(t) = f(x(t), t)$, and a tube $G(t) = \{x(t) \mid g(x(t), t) \leq 0\}$. Consider the constraint system:

\[
\begin{align*}
(i) & \quad \frac{\partial g_i(x, t)}{\partial x} \cdot f(x, t) + \frac{\partial g_i(x, t)}{\partial t} \geq 0 \\
(ii) & \quad g_i(x, t) = 0 \\
(iii) & \quad g(x, t) \leq 0 
\end{align*}
\]

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

The constraint system (1) describes the subset of $S_f$ trajectories that escape from $G(t)$. If this subset is empty, it means that $G(t)$ is a capture tube.

In [9], 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. 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:

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

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 $S_f$ be a dynamic system defined by $\dot{x}(t) = f(x(t), t)$, and a candidate tube $G(t) = \{x(t) \mid g(x(t), t) \leq 0\}$.

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

- differential constraint: $\dot{x}(t) = f(x(t), t)$
- cross out constraint: $\exists t_0, x(t_0)$ satisfies (1)
- escape constraint: $\forall t \in [t_0, t_f], g(x(t), t) > 0$

The constraints model the fact that the system can escape from $G(t)$ “for ever”, i.e. cannot go back in $G(t)$ before $t_f$. If $N$ is inconsistent, then it proves that $G(t)$ 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 [9].
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 $G(t)$ "for ever", i.e. at a time greater than $t_0$ and remaining outside $G(t)$ 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 $G(t)$ and precision parameters on the state variables ($\epsilon_{start}$, $\epsilon_{min}$) and on the time ($timestep$). They are detailed further.

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. The tube can become empty (and $ContractionResult = in$) if $Contraction$ could prove that the tube is entirely inside $G(t)$ at an instant between $t_0$ and $t_f$ (see Lines 16–18). A second case occurs when tube has been detected outside $G(t)$ 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 more information at this time (cross out conditions hold). Tube bisection is not achieved if 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 $G(t)$ is a quasi invariant tube for the system $S$.

![Figure 3](image-url) Example of a branching and contraction of the cross out conditions

...Figure 4 shows how the simulation works for different boxes of "QCTcandidates".

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 constraints stating that the trajectories in tube cross $G(t)$ out (see Fig. 3). It calls the state-of-the-art contractor hC4 [ref HC4] or 3bcid [ref 3bcid] on the cross out constraint subsystem (see Section ??? describing the experiments).
Algorithm 1 Branch and contract

1 Input \((G(t), \text{initTube}, t_0, t_f, \text{timestep}, \epsilon_{\text{start}}, \epsilon_{\text{min}})\)
2 Output (OutList : list of solution tubes ; UndeterminedList : list of 'small' tubes still undetermined)
3 tubes ← \{initTube\}
4 while (tubes ≠ \emptyset) do
  5 tube ← Pop(tubes)
  6 (ContractionResult, tube) ← Contraction(tube, S, G(t), t_0, t_f, \text{timestep}, \epsilon_{\text{start}})
  7 if (ContractionResult = out) then
    8 OutList ← OutList ∪ \{tube\}
  else if (ContractionResult = undetermined) then
    9 if Diam(tube) ≤ \epsilon_{\text{min}} then
      10 UndeterminedList ← UndeterminedList ∪ \{tube\}
    else
      11 (tube_{left}, tube_{right}) ← Bisect(tube, bisectionStrategy)
      12 tubes ← \{tube_{left}\} ∪ \{tube_{right}\} ∪ tubes
    end
  else
    17 /* ContractionResult = in: Nothing to do : tube is discarded because its trajectories all enter inside \(G(t)\) at an instant in \([t_0, t_f]\) */
  end
19 end

With the call to 0DEEvalContraction (Line 6), we then proceed with the contraction of the differential (ODE) constraint and the espace constraint. Note that this contraction procedure is run only under a given level of the search tree, where the tube diameter is lower than a user parameter \(\epsilon_{\text{start}}\). 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, G(t), tube, t_0, t_f, \text{timestep}, \epsilon_{\text{start}})
2 tube ← CrossOutContraction(tube, S, G(t))
3 if (tube = \emptyset) then
  4 ContractionResult ← in
4 else if (Diam(tube) < \epsilon_{\text{start}}) then
  6 ContractionResult ← 0DEEvalContraction(S, tube, G(t), t_0, t_f, \text{timestep})
else
  8 ContractionResult ← undetermined
end
10 return (ContractionResult, tube)
11 end
Figure 4 Examples of simulation. The green box returns to \( C(t) \). The yellow box of the cross out conditions should be bisected to compute another simulation to conclude if the trajectory returns to \( C(t) \) or not. The pink box is a solution for the CSP.

4.2 Differential contraction

White box differential contractors, e.g. the ctcDeriv and ctcEval contractors available in the TUBEX/CODAC free library [20], 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 [17] or CAPD [10], to benefit from its optimized internal representations. The corresponding method is described in Algorithm 3.

Algorithm 3 Function ODEEvalContraction called by Algorithm 2

1 Function ODEEvalContraction(\( S, \) tube, \( C(t), t_0, t_f, \) timestep)
2 \( t_i \leftarrow t_0 \)
3 \( t_{out} \leftarrow \infty \)
4 repeat
5 \( (slice, t_{i+1}) \leftarrow \) GISimulation(\( S, \) tube\( (t_i), t_i, t_f \))
6 \( (\)ContractionResult, \( t_{out} \)\) \leftarrow \) GI_Eval(\( slice, G(t), \) timestep, \( t_i, t_{i+1}, t_{out} \))
7 tube[\( t_i, t_{i+1} \)] \leftarrow tube[\( t_i, t_{i+1} \)] \cap slice
8 \( t_i \leftarrow t_{i+1} \)
9 until \( t_i = t_f \) or (ContractionResult = in)
10 if ContractionResult = in or \( t_{out} \neq \infty \) then
11 \| return ContractionResult
12 else
13 \| return undetermined
14 end
15 end

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 GI
A. Bedouhene et al.

The solver has built 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 \(t_{\text{timestep}}\) and tests whether tube during the subslice studied satisfies the escape (from \(G(t)\)) constraint or not. In the latter case, the integration is interrupted (Algorithm 3 stops) and ContractionResult is set to \(\text{in}\). The whole tube is rejected. If a subslice satisfies the escape constraint, \(t_{\text{out}}\) is used to memorize the first instant where it occurs. If \(t_{\text{out}} = \infty\), then \(t_{\text{out}}\) is set to \(t_i\). If a subsequent subslice evaluation does not return \(\text{out}\), then \(t_{\text{out}}\) is set back to \(\infty\). Indeed, recall that a solution tube must satisfy the escape constraint in all times from \(t_{\text{out}}\) to \(t_f\). When \(t_f\) is reached, only two cases are still possible. Either tube has escaped from \(G(t)\) at \(t_{\text{out}}\) until \(t_f\) (a solution), or tube has intersected \(G(t)\) at some instants, including \(t_f\). In that case, we cannot conclude and the result of the contraction will be \(\text{undetermined}\).

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 \(\text{undetermined}\) result since the algorithm is not able to conclude if the tube goes inside \(G(t)\) or not. The choice of \(\epsilon_{\text{start}}\) has a significant impact on the frequency of this “pathological” case (see experiments).

5 Experiments

The current section presents some results provided by "Bubbibex", an implementation of the algorithm [réf algo 1]. "Bubbibex" is implemented in C++. It uses the IBEX library [4] with the hc4 [2] and 3bcid [13, 22] contractors for propagating the cross out conditions constraints. It also uses the CAPD/DynSys library for the differential contractor based on guaranteed integration [10] and Tubex/CODAC library for tube structures [20].

Experiments are carried out using an Intel(R) Xeon(R) CPU E3-1225 V2 at 3.20GHz. Each experiment, see table [1], is studied in order to highlight the different responses of the solver to some parameters (\(\epsilon_{\text{start}}, \) bubble radius...) of the algorithm or to the nature of the problem itself. These responses are measured with the time execution of each experiment (CPU-Time) and the number and nature of the computed tubes corresponding to the leaves of the search tree: "In" for returning tubes, "Und" for undetermined tubes and "Out" for tubes staying out at \(t_f\).

The simulation time of each experiment at most \(t_f = 100\) with \(t_{\text{timestep}} = 0.01\).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Time dependent</th>
<th>State variables</th>
<th>Bubble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pendulum</td>
<td>Non Linear</td>
<td>no</td>
<td>2</td>
<td>Static</td>
</tr>
<tr>
<td>Tracking</td>
<td>Linear</td>
<td>yes</td>
<td>2 and 3</td>
<td>Static and moving</td>
</tr>
<tr>
<td>Dubins car</td>
<td>Non Linear</td>
<td>yes</td>
<td>2</td>
<td>Moving</td>
</tr>
<tr>
<td>Pursuit game</td>
<td>Non Linear</td>
<td>yes</td>
<td>3 and 5</td>
<td>Moving</td>
</tr>
</tbody>
</table>

5.1 Pendulum (Autonomous system)

\[
P : \begin{cases} 
    \dot{x} = y \\
    \dot{y} = -\sin(x) - 0.15.y 
\end{cases}
\]

Let \(P\) a dynamical system describing the motion of a pendulum such that, \(x\) is the angular position and \(y\) is the angular velocity. We want to find a quasi capture tube for the system 2.

Parameters of the experiment
23:10 Interval CP for Quasi Capture Tube Validation

<table>
<thead>
<tr>
<th>First gate</th>
<th>Bubble</th>
<th>$r_0$</th>
<th>Observed parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y \in [-10, 10]$</td>
<td>$x^2 + y^2 - r_0^2 \leq 0$</td>
<td>1</td>
<td>$\epsilon_{\text{start}}$</td>
</tr>
</tbody>
</table>

Table 2: When $\epsilon_{\text{start}} = \{1,1\}$ (line 1) the differential contractor is not able to successfully contract the tube, this is due to a large initial condition for the guaranteed integration which failed to compute a solution leading the solver to bisect the initial gate of the tube before reaching the right precision. Having a good intuition on the parameter $\epsilon_{\text{start}}$ (line 2) can improve the efficiency of the method. The CSP has no solution, $g(x, y)$ is a quasi capture tube.

<table>
<thead>
<tr>
<th>$\epsilon_{\text{start}}$</th>
<th>$\epsilon_{\text{min}}$</th>
<th>In</th>
<th>Und</th>
<th>Out</th>
<th>CPU-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1,1}$</td>
<td>${0.1,0.1}$</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>72.171</td>
</tr>
<tr>
<td>${0.5,0.5}$</td>
<td>${0.1,0.1}$</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.00734</td>
</tr>
</tbody>
</table>

5.2 Linear tracking system

Let the linear dynamical system:

$$\dot{x}(t) = A(x(t) - T(t))$$ (3)

With $x = [x_1, \ldots, x_n]^T$ the tracking system and $T(t)$ the target.

We want to study the stability of the system (3) by finding a quasi capture tube. We will study two cases for the system (3), one with a static bubble centered on the origin, and the other one with a moving bubble centered on the target.

Parameters of the experiment:

<table>
<thead>
<tr>
<th>First gate</th>
<th>Bubble</th>
<th>$r_0$</th>
<th>Observed parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1, \ldots, x_n \in [-10, 10]$</td>
<td>$(x_1)^2 + \cdots + (x_n)^2 - r_0^2 \leq 0$</td>
<td>2</td>
<td>Dim/Bubble</td>
</tr>
<tr>
<td>$x_1, \ldots, x_n \in [-10, 10]$</td>
<td>$(x_1 - T_1(t))^2 + \cdots + (x_n - T_n(t))^2 - r_0^2 \leq 0$</td>
<td>2</td>
<td>Dim/Bubble</td>
</tr>
</tbody>
</table>

2D and 3D tracking systems

Consider for the 2D linear system:

$$n = 2: \quad A = \begin{bmatrix} 1 & 3 \\ -3 & -2 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos t \\ \sin 2t \end{bmatrix}$$ (4)

And for the 3D linear system:

$$n = 3, \quad A = \begin{bmatrix} 1 & 3 & 0 \\ -3 & -2 & -1 \\ 0 & 1 & -3 \end{bmatrix}, \quad T(t) = \begin{bmatrix} \cos t \\ \cos t \sin 2t \\ -\sin t \sin 2t \end{bmatrix}$$ (5)

<table>
<thead>
<tr>
<th>$\epsilon_{\text{start}}$</th>
<th>$\epsilon_{\text{min}}$</th>
<th>${t_0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1,1}$</td>
<td>${0.05}$</td>
<td></td>
</tr>
<tr>
<td>${0.1,0.1}$</td>
<td>${0.01}$</td>
<td></td>
</tr>
<tr>
<td>${1,1,1}$</td>
<td>${0.05}$</td>
<td></td>
</tr>
<tr>
<td>${0.1,0.1,0.1}$</td>
<td>${0.01}$</td>
<td></td>
</tr>
</tbody>
</table>

Both targets, in the 2D linear system and the 3D linear system, have periodic pattern movement and their period is $2\pi$. We can then reduce 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 3 we can conclude that both bubbles are quasi capture tubes for the system (3).
Table 3 Results for both systems (2D and 3D) and both bubbles (static and moving).

<table>
<thead>
<tr>
<th>Dim</th>
<th>Bubble</th>
<th>( \epsilon_{\text{start}} )</th>
<th>( \epsilon_{\text{min}} )</th>
<th>In</th>
<th>Und</th>
<th>Out</th>
<th>CPU-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Static</td>
<td>{1,1,0,0.05}</td>
<td>{0.1,0.1,1.0,0.01}</td>
<td>370</td>
<td>0</td>
<td>0</td>
<td>1.20</td>
</tr>
<tr>
<td>2D</td>
<td>Moving</td>
<td>{1,1,0.05}</td>
<td>{0.1,0.1,1.0,0.01}</td>
<td>1021</td>
<td>0</td>
<td>0</td>
<td>1.65</td>
</tr>
<tr>
<td>3D</td>
<td>Static</td>
<td>{1,1,1,0.05}</td>
<td>{0.1,0.1,0.1,0.01}</td>
<td>3290</td>
<td>0</td>
<td>0</td>
<td>7.10</td>
</tr>
<tr>
<td>3D</td>
<td>Moving</td>
<td>{1,1,1,0.05}</td>
<td>{0.1,0.1,0.1,0.01}</td>
<td>4040</td>
<td>0</td>
<td>0</td>
<td>11.94</td>
</tr>
</tbody>
</table>

Figure 5 Sample of tubes leaving the static bubble of the 3D linear tracking system. Upper left: Gates of the cross out condition on a sphere of radius \( r_0 = 2 \) enveloping the target (going from red at \( t = 0 \) to white at \( t = 2\pi \)). Upper right and lower left: Tubes entering almost immediately to the sphere. Lower right: Tube going far away from the sphere before one first unsuccessful landing.

5.3 Dubins car (Time dependent system)

\[
R: \begin{cases} 
\dot{x} = u_1 \\
\dot{y} = u_2 \\
\dot{\theta} = -\theta 
\end{cases} \tag{6}
\]

Let \( R \) a robot described by the dynamical system (6) such that, \((x, y)\) is the position, \( \theta \) the heading and \( u_1 = -x + t, \ u_2 = -y \) the controllers.

We want the robot to stay inside a time moving bubble.

We also want to make sure that the initial heading is setup correctly, so we add the following constraint on the heading with the constraints of the cross out condition:

\[
h(x, y, t) = (\cos(\theta) - 1)^2 + (\sin(\theta))^2 - \epsilon_\theta \leq 0
\]

Parameters of the experiment:

<table>
<thead>
<tr>
<th>First gate</th>
<th>Bubble</th>
<th>( \epsilon_\theta )</th>
<th>Observed parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, y \in [-100, 100] )</td>
<td>((x - t)^2 + (y)^2 - r_0 \leq 0 )</td>
<td>0.2</td>
<td>( r_0 )</td>
</tr>
</tbody>
</table>

For bubbles with radius \( r_0 > 1 \) the solver is able to verify that they are capture tubes (the set of the cross-out condition is empty).

The table 4 depicts the results of bubbles with radius \( r_0 = 1, \ r_0 = 0.9 \) and a time dependent radius \( r_0 = \frac{1}{\sqrt{5}} (1 + t) \). For instance, we are able to prove that for \( r_0 = 0.9 \) the
bubble is not a quasi capture tube, but we are not able to conclude for a small $\epsilon_{\text{min}}$. On the other hand, the bubble with radius $r_0 = \frac{1}{\sqrt{5}}(1 + t)$ is a quasi capture tube.

Table 4 Results for $r_0 = 1$, $r_0 = 0.9$ and $r_0 = \frac{1}{\sqrt{5}}(1 + t)$.

<table>
<thead>
<tr>
<th>$r_0$</th>
<th>$[t_0]$</th>
<th>$\epsilon_{\text{start}}$</th>
<th>$\epsilon_{\text{min}}$</th>
<th>In Und Out</th>
<th>CPU-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>{1,1,1}</td>
<td>{0.1,0.1,0.1}</td>
<td>0</td>
<td>256</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>{1,1,1}</td>
<td>{1e-4,1e-4,1e-4}</td>
<td>0</td>
<td>32764</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td>{1,1,1}</td>
<td>{0.1,0.1,0.1}</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\frac{1}{\sqrt{5}}(1+t)$</td>
<td>0,100</td>
<td>{1,1,1,0.1}</td>
<td>{0.1,0.1,0.1,0.01}</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

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 on the "pursuit evasion" game, one on the plane, and the other one in the 3d-space. The evader (E) will be at the center of a moving 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

Let the pursuer P and the evader E:

$$\begin{align*}
P : & \begin{cases}
\dot{x} = u_1 \cos(\theta) \\
\dot{y} = u_1 \sin(\theta) \\
\dot{\theta} = u_2
\end{cases} \\
E : & \begin{cases}
x_d = f(x_d) = v.t \\
y_d = f(y_d) = \sin(\rho t)
\end{cases}
\end{align*}$$  \hspace{1cm} \text{(7)}$$

Where:

- $x$ and $y$ its position and $\theta$ its heading.
- 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 $\theta_d = \tan(2(n))$ and $n$ is defined as follows:

$$n = \left[ \begin{array}{c} x_d - x \\
y_d - y 
\end{array} \right] + dt \left[ \begin{array}{c} \dot{x}_d \\
\dot{y}_d \end{array} \right]$$

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

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

Constants: $K = 1$, $v = 7$, $\rho = 1$, $r = 1$, $dt = 1$, $\epsilon_\theta = 0.02$

Parameters:

<table>
<thead>
<tr>
<th>First gate</th>
<th>Bubble</th>
<th>$r_0$</th>
<th>Observed parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y \in [-10, 10], \theta \in [0, 2\pi]$</td>
<td>$(x - x_d)^2 + (y - y_d)^2 - r_0^2 = 0$</td>
<td>1</td>
<td>$\epsilon_h = \epsilon_\theta$</td>
</tr>
</tbody>
</table>

Precision:

<table>
<thead>
<tr>
<th>$\epsilon_{\text{start}}$</th>
<th>$[t_0]$</th>
<th>$\epsilon_{\text{min}}$</th>
<th>$\epsilon_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0.1,0.1,0.1}</td>
<td>{0.05}</td>
<td>{0.01,0.01,0.005}</td>
<td>{0.005}</td>
</tr>
</tbody>
</table>
Pursuit Evasion game in the 3D-space

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

$$\begin{align*}
\dot{x} &= u_1 \cos(\theta) \cos(\psi) \\
\dot{y} &= u_1 \cos(\theta) \sin(\psi) \\
\dot{z} &= u_2 \\
\dot{\psi} &= u_3
\end{align*}$$

$$P : \left\{ \begin{array}{ll}
x_d &= f(x_d) = v \cdot w \cdot t \\
y_d &= f(y_d) = v \cdot w \cdot \sin(w \cdot t) \\
z_d &= f(z_d) = -v \cdot w \cdot \cos(w \cdot t)
\end{array} \right. \quad (8)$$

Where:

$x$, $y$, and $z$ its position, $\psi$ its circular rotation speed and $\theta$ its vertical rotation speed. The controls $u_1 = ||n||$, $u_2 = \psi - \psi_d$ and $u_3 = \theta - \theta_d$.

We add constraints on the circular and vertical rotations of the pursuer:

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

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

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

Parameters:

<table>
<thead>
<tr>
<th>First gate</th>
<th>Bubble</th>
<th>$r_0$</th>
<th>Observed parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z \in [-10, 10]$</td>
<td>$\bar{r}_0$</td>
<td>$1$</td>
<td>$\epsilon_\theta = \epsilon_\psi = \epsilon_\theta$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\theta, \psi \in [0, 2\pi]$</th>
<th>$\epsilon_{\text{start}}$</th>
<th>$[0.05]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_{\text{end}}$</td>
<td>$0.01, 0.01, 0.005, 0.005$</td>
<td>$[0.005]$</td>
</tr>
</tbody>
</table>

Pursuit Evasion game Results

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

As we increase in the complexity of the problem (number of the state variables, non linearity, stiffness...), the solver faces some difficulties. We can see in tables 5 and 6 how varies the number of tubes computed by the solver in order to validate a quasi capture tube compared to the previous experiments. The number of these tubes can be drastically lowered by using small parameters $\epsilon_\theta$ (resp. $\epsilon_\psi, \epsilon_\theta$) to restrict the initial heading (resp. circular and vertical rotations) of the pursuer.

Table 5 Results of the pursuit game on the plane show that with a small parameter $\epsilon_\theta$ we can validate the quasi capture tube on the whole period of the evader.
Table 6 Even for small parameters \((\epsilon_\psi, \epsilon_\theta)\), one tenth of the period for \([t_0] \) requires a huge CPU-Time execution. On the other hand, the quasi capture tube is validated.

<table>
<thead>
<tr>
<th>([t_0])</th>
<th>(\epsilon_h)</th>
<th>(\epsilon_\text{start})</th>
<th>(\epsilon_\text{min})</th>
<th>In</th>
<th>Und</th>
<th>Out</th>
<th>CPU-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.045</td>
<td>{0.1,0.1,0.1,0.05,0.05}</td>
<td>{0.01,0.01,0.01,0.005,0.005}</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>4.91</td>
</tr>
<tr>
<td>([0, \pi/5])</td>
<td>0.045</td>
<td>{0.1,0.1,0.1,0.05,0.05}</td>
<td>{0.01,0.01,0.01,0.005,0.005}</td>
<td>115301</td>
<td>0</td>
<td>0</td>
<td>44084.44</td>
</tr>
</tbody>
</table>

Figure 6 Pursuit evasion game in 3d-space. Gates generated by the contraction of the cross out condition constraint at \([t_0] = 0\) around the sphere of radius \(r_0 = 1\) centered on the position of the evader (in red) at \([t_0] = 0\). We can notice how the number of gates varies for different values for \(\epsilon_h\). Upper left: \(\epsilon_h = 0.05\). Upper right: \(\epsilon_h = 0.0625\). Lower left: \(\epsilon_h = 0.08\). Lower right: \(\epsilon_h = 0.1\).

References


