Guaranteed Computation of Robot Trajectories

Simon Rohou[a], Luc Jaulin[a], Lyudmila Mihaylova[b], Fabrice Le Bars[a], Sandor M. Veres[b]

[a] *ENSTA Bretagne, Lab-STICC, UMR CNRS 6285, Brest, France*
[b] *The University of Sheffield, Department of Automatic Control and Systems Engineering, Sheffield, United Kingdom*

Abstract

This paper proposes a new method for guaranteed integration of state equations. Within this framework, the variables of interest are trajectories submitted to both arithmetic and differential equations. The approach consists in formalizing a problem thanks to a constraint network and then apply these constraints to sets of trajectories. The contribution of the paper is to provide a reliable framework to enclose the solutions of these differential equations. Its use is shown to be simple, more general and more competitive than existing approaches dealing with guaranteed integration, especially when applied to mobile robotics. The flexibility of the developed framework allows to deal with non-linear differential equations or even differential inclusions built from datasets, while considering observations of the states of interest. An illustration of this method is given over several examples with mobile robots.

Keywords: guaranteed integration, tube programming, mobile robotics, constraints, contractors, ODE

1. Introduction

In this paper, we consider the problem of *guaranteed integration* of dynamical systems (see *e.g.* [1]) of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{n}(t) \tag{1}$$

where $t \in \mathbb{R}$ is the time, $\mathbf{x}(t)$ is the state vector, $\mathbf{n}(t)$ is the noise vector, assumed to belong to a known box $[\mathbf{n}]$, and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is the *evolution* function [2]. One of the main motivations of guaranteed integration methods is to develop reliable cyber-physical systems such as Acumen [3] for dynamical systems simulation and verification. From a mathematical point of view, our problem corresponds to the area of *interval integration* [4, 5], the aim of which is to compute a guaranteed enclosure of an Initial Value Problem (IVP). Efficient libraries for interval integration of differential equations are available, such as Cosy [6], Vnode [7], DynIbex [8] or Capd [9]. These libraries are used in robotics and automatic control to verify dynamical properties of non-linear systems [10], to compute reachable sets [11, 12] or for state estimation [13]. They are also used by mathematicians to prove conjectures [14, 15].

From an initial box $[\mathbf{x}](0)$, representing a bounded initial state at time $t = 0$, guaranteed integration provides a set of techniques to compute a box-valued function $[\mathbf{x}](t)$ (or *tube*) containing all feasible solutions of an Ordinary Differential Equation (ODE) or a differential inclusion. Existing methods are based on interval extensions of the Euler [4], Runge-Kutta [8] or Taylor [6] integrations and the application of the Picard-Lindelf Existence Theorem.

This paper studies a constraint-based approach [16, 17] for guaranteed interval integration. Our approach consists in rephrasing a problem of *trajectory integration* into a problem of *constraint satisfaction*. To achieve this goal, we introduce a new tool, a dedicated *contractor*, which allows us to consider constraints defined by a differential inclusion, in a guaranteed way. We show that the genericity of the method allows to deal with a wide range of problems involving differential equations. The user has to define a dynamical system with a set of constraints and let the *contractors* approximate the envelope containing the set of solutions. Meanwhile, other kinds of constraints can be defined in the system, such as state observations, and easily considered by our dynamical contractor. To our knowledge, such simple, generic and reliable approach has not been proposed so far.

The paper is organized as follows. Section 2 presents the principle of constraint propagation and how it can be related to sets of trajectories. It relies on existing works, that will serve as the basis of Section 3 in which we provide a new approach to consider dynamical systems and solve an initial value problem. The simplicity of the approach is highlighted in Sections 4 and 5 that illustrate, respectively, the tool with robotics simulations and state estimations based on real datasets. A comparison with Capd [9] (which can be considered as one of the most efficient libraries in this field) shows that our method can also be considered as competitive, at least for robotics applications. Sections 6 and 7 conclude the paper and present the numerical libraries used during this work.

2. Constraint Propagation over Trajectories

Subsection 2.1 recalls the principle of constraint propagation [16, 17] that will be used later to formalize a prob-

lem of interval integration and then state estimations in robotics applications. Dynamical systems lead to constraints to be applied on sets of trajectories. To this end, a tool such as a *tube* can be used to enclose the domain of the feasible solution set. Its description is given in Subsection 2.2.

2.1. Constraint Satisfaction Problems

CSP. In numerical contexts, problems of control, state estimation and robotics can be described as *Constraint Satisfaction Problems* (CSP) in which variables must satisfy a set of rules or facts, called constraints, over domains defining a range of feasible values. Links between the constraints define a *Constraint Network* [18] involving variables $\{x_1, \ldots, x_n\}$, constraints $\{\mathcal{L}_1, \ldots, \mathcal{L}_m\}$ and domains $\{\mathbb{X}_1, \ldots, \mathbb{X}_n\}$ containing the x_i's. The variables x_i can be symbols, real numbers [19], vectors of $\mathbb{R}^n$, and sometimes trajectories [20, 21]. The constraints can be non-linear equations between the variables, such as $x_3 = \cos(x_1 + \exp(x_2))$, or differential equations: $\dot{x}_3 \cdot \ddot{x}_1 + \exp(x_2) = 0$. Domains are intervals, boxes [22], polytopes [23, 24] or tubes [25, 20] when dealing with trajectories.

Contractors. A constraint $\mathcal{L}$ can be applied on a box $[\mathbf{x}] \in \mathbb{IR}^n$ with the help of a contractor $\mathcal{C}$. The box $[\mathbf{x}]$, also called *interval-vector*, is a closed and connected subset of $\mathbb{R}^n$ and belongs to the set of n-dimensional boxes denoted $\mathbb{IR}^n$. Formally, a contractor $\mathcal{C}$ associated to the constraint $\mathcal{L}$ is an operator $\mathbb{IR}^n \to \mathbb{IR}^n$ that returns a box $\mathcal{C}([\mathbf{x}]) \subseteq [\mathbf{x}]$ without removing any vector consistent with $\mathcal{L}$ [22, 26]. Constructing a store of contractors such as $\mathcal{C}_+$, $\mathcal{C}_{\sin}$, $\mathcal{C}_{\exp}$ associated to primitive equations such as $z = x + y$, $y = \sin(x)$, $y = \exp(x)$ has been the subject of much work.

Decomposition. Problems involving complex equations can be broken down into a set of primitive equations. Here, *primitive* means that the constraint cannot be decomposed anymore and that the related contractor is available in the collection of contractors, thus allowing to deal with a wide range of problems. This can also be applied on trajectories. For instance, the differential equation $\ddot{x}(\cdot) \cdot \dot{x}(\cdot) = x(\cdot)$ can be decomposed into:

$$\left\{ \begin{array}{rcl} \dot{x} & = & b \\ \dot{b} & = & a \\ x & = & a \cdot b \end{array} \right. \tag{2}$$

Combining primitive contractors leads to a complex contractor that still provides reliable results.

Propagation. When working with finite domains, a propagation technique can be used to solve a problem. The process is run up to a fixed point when domains $\mathbb{X}_i$ cannot be reduced anymore.

Our goal is to consider trajectories as variables and to implement contractors to reduce their domains given a differential constraint. This will be done with the help of tubes.

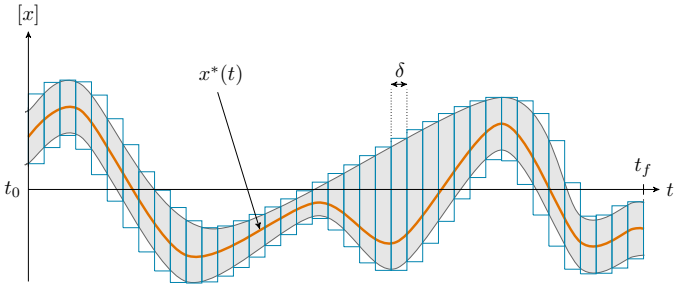


Figure 1: A tube $[x](\cdot)$ represented by a set of slices. This representation can be used to enclose signals such as $x^*(\cdot)$.

2.2. Tubes: envelopes of feasible trajectories

Definition. A tube is defined [27, 28] as an envelope enclosing an uncertain trajectory $\mathbf{x}(\cdot) : \mathbb{R} \to \mathbb{R}^n$. In this paper, we will use the definition given in [20, 21] where a tube $[\mathbf{x}](\cdot)$ is an interval of two trajectories $[\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$ such that $\forall t,\ \mathbf{x}^-(t) \leqslant \mathbf{x}^+(t)$. A trajectory $\mathbf{x}(\cdot)$ belongs to the tube $[\mathbf{x}](\cdot)$ if $\forall t,\ \mathbf{x}(t) \in [\mathbf{x}](t)$. Figure 1 illustrates a tube implemented with a set of boxes. This sliced implementation is detailed hereinafter.

Arithmetics on tubes. Consider two tubes $[x](\cdot)$ and $[y](\cdot)$ and an operator $\diamond \in \{+, -, \cdot, /\}$. We define $[x](\cdot) \diamond [y](\cdot)$ as the smallest tube (with respect to the inclusion) containing all feasible values for $x(\cdot) \diamond y(\cdot)$, assuming that $x(\cdot) \in [x](\cdot)$ and $y(\cdot) \in [y](\cdot)$. This definition is an extension to trajectories of the interval arithmetic proposed by Moore [4]. If f is an elementary function such as sin, cos, ..., we define $f\big([x](\cdot)\big)$ as the smallest tube containing all feasible values for $f\big(x(\cdot)\big)$, $x(\cdot) \in [x](\cdot)$.

Integrals of tubes. In the same way, the integral of a tube [29] is defined from t_1 to t_2 as the smallest box containing all feasible integrals:

$$\int_{t_1}^{t_2} [\mathbf{x}](\tau) d\tau = \left\{ \int_{t_1}^{t_2} \mathbf{x}(\tau) d\tau \mid \mathbf{x}(\cdot) \in [\mathbf{x}](\cdot) \right\}. \tag{3}$$

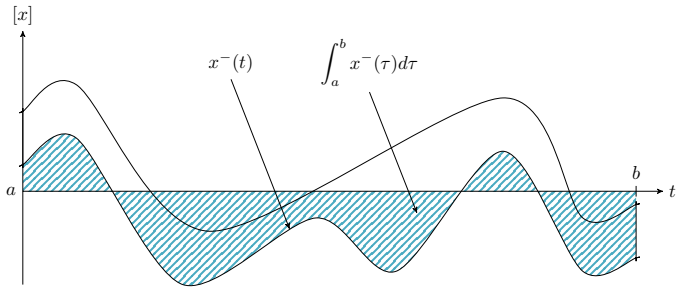
From the monotonicity of the integral operator, we can deduce:

$$\int_{t_1}^{t_2} [\mathbf{x}](\tau) d\tau = \left[\int_{t_1}^{t_2} \mathbf{x}^-(\tau) d\tau, \int_{t_1}^{t_2} \mathbf{x}^+(\tau) d\tau \right], \tag{4}$$

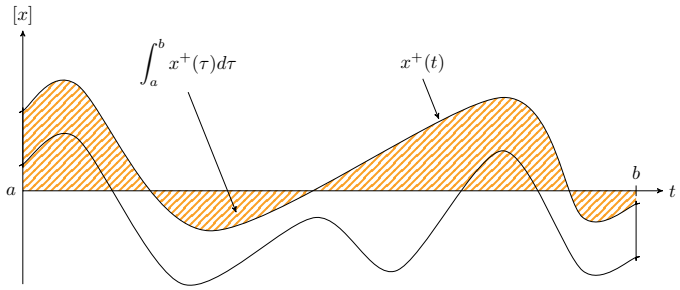
where $\mathbf{x}^-(\cdot)$ and $\mathbf{x}^+(\cdot)$ are the lower and upper bounds of tube $[\mathbf{x}](\cdot) = [\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$. The computed integral is a box with lower and upper bounds shown on Figure 2. For efficiency purposes, the *interval primitive* of a tube defined by $\int_0^t [\mathbf{x}](\tau) d\tau$ can be computed once, giving a primitive tube.

Simple example. Figures 3a–3b present two scalar tubes $[x](\cdot)$ and $[y](\cdot)$. The tube arithmetic makes it possible to compute the following tubes, as depicted by Figures 3c–3e:

$$\begin{array}{rcl} [a](\cdot) & = & [x](\cdot) + [y](\cdot) \\ [b](\cdot) & = & \sin([x](\cdot)) \\ [c](\cdot) & = & \int_0 [x](\tau)\, d\tau \end{array} \tag{5}$$



(a) hatched part depicts the lower bound of $\int_a^b [x](\tau)d\tau$



(b) hatched part depicts the upper bound of $\int_a^b [x](\tau)d\tau$

Figure 2: Lower and upper bounds of a tube's integral

Contractors for tubes. The contractors recalled in Subsection 2.1 can be applied on sets of trajectories, thus allowing constraints over time such as $a(t) = x(t) + y(t)$ or $b(t) = \sin(x(t))$. A *tube contractor* has been defined in [21] and is recalled here. A contractor applied on a tube $[a](\cdot)$ aims at removing unfeasible trajectories according to a given constraint $\mathcal{L}$:

$$[a](\cdot) \xrightarrow{\mathcal{C}_\mathcal{L}} [b](\cdot). \tag{6}$$

The output of the contractor $\mathcal{C}_\mathcal{L}$ is the tube $[b](\cdot)$ such that:

$$\forall t,\ [b](t) \subseteq [a](t), \qquad \text{(contraction)} \tag{7}$$

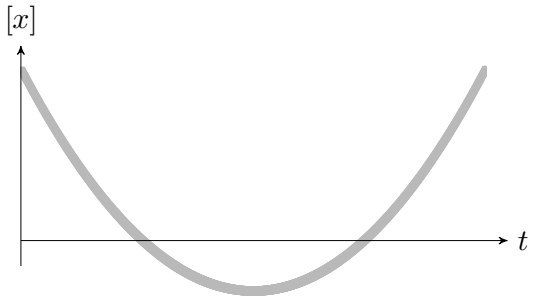
$$\left(\begin{array}{c} \mathcal{L}\big(a(\cdot)\big) \\ a(\cdot) \in [a](\cdot) \end{array} \right) \Longrightarrow a(\cdot) \in [b](\cdot). \quad \text{(completeness)} \tag{8}$$

For instance, the minimal contractor $\mathcal{C}_+$ associated with the constraint $a(\cdot) = x(\cdot) + y(\cdot)$ is:

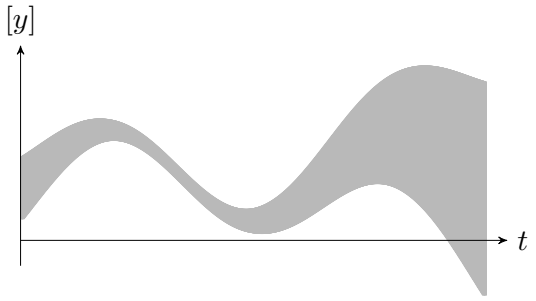
$$\left(\begin{array}{c} [a](\cdot) \\ [x](\cdot) \\ [y](\cdot) \end{array} \right) \mapsto \left(\begin{array}{c} [a](\cdot) \cap ([x](\cdot) + [y](\cdot)) \\ [x](\cdot) \cap ([a](\cdot) - [y](\cdot)) \\ [y](\cdot) \cap ([a](\cdot) - [x](\cdot)) \end{array} \right). \tag{9}$$

In this way, information on either $[a](\cdot)$, $[x](\cdot)$ or $[y](\cdot)$ can be propagated to the other tubes.

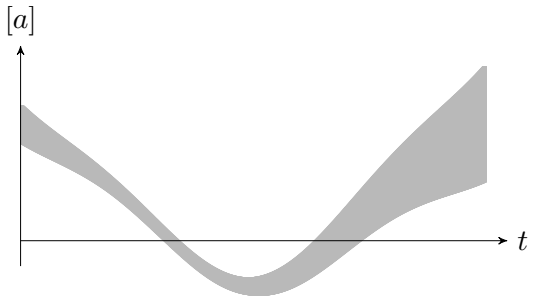
Implementation. There are several ways to implement a tube. Our choice is to build it with a set of boxes representing slices of identical width. Figure 1 illustrates such implementation with a list of boxes, while keeping enclosed an unknown trajectory $x^*(t)$:$\mathbb{R} \to \mathbb{R}$. More precisely, a tube $[\mathbf{x}](t)$, with a sampling time $\delta > 0$, is a box-valued function which is constant for all t inside intervals $[k\delta, k\delta + \delta]$, $k \in \mathbb{Z}$. The box $[k\delta, k\delta + \delta] \times [\mathbf{x}](t_k)$,



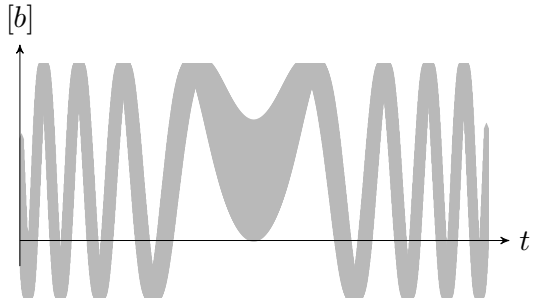
(a) Tube $[x](t) = t^2 + [e]$ where $[e] \in \mathbb{IR}$ is an arbitrary interval corresponding to tube's constant thickness.



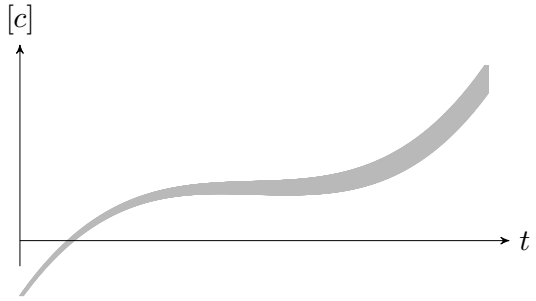
(b) Tube $[y](t) = -cos(t) + h(t) \cdot [e]$ where $h : \mathbb{R} \to \mathbb{R}$ is an arbitrary function depicting a thickness changeover.



(c) Tube $[a](t) = [x](t) + [y](t)$



(d) Tube $[b](t) = \sin([x](t))$



(e) Tube $[c](t) = \int_0^t [x](\tau)d\tau$

Figure 3: Tubes as described by Equation (5) with tube arithmetics. Note that the vertical scales of these figures vary for full display.

with $t_k \in [k\delta, k\delta + \delta]$ is called the k^{th} *slice* of the tube $[\mathbf{x}](\cdot)$ and is denoted by $[\mathbf{x}](k)$. The resulting approximation of a tube encloses $[\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$ inside an interval of step functions $[\underline{\mathbf{x}^-}(\cdot), \overline{\mathbf{x}^+}(\cdot)]$ such that:

$$\forall t,\ \underline{\mathbf{x}^-}(t) \leqslant \mathbf{x}^-(t) \leqslant \mathbf{x}^+(t) \leqslant \overline{\mathbf{x}^+}(t). \tag{10}$$

Such implementation then takes rigorously into account floating point precision when building a tube, thanks to reliable numerical libraries such as [30].

Further computations involving $[\mathbf{x}](\cdot)$ will be based on its slices, thus giving an outer approximation of the solution set. For instance, the lower bound of the integral of a tube, defined in Equation (4), is simply computed as the signed area of the region in the tx-plane that is bounded by the graph of $\underline{\mathbf{x}^-}(t)$ and the t-axis, as pictured in Figure 4. The lower slice width δ, the higher the precision of the approximation.

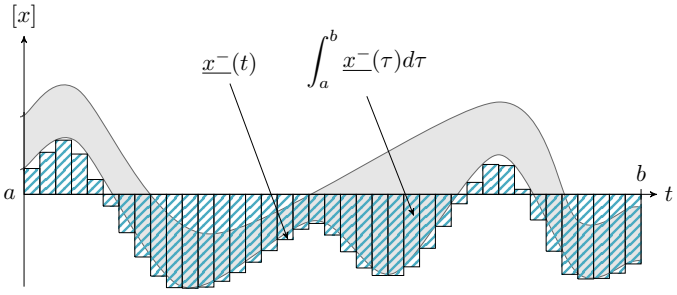


Figure 4: Outer approximation of the lower bound of $\int_a^b [x](\tau)d\tau$.

3. Reliable Integration

This section provides a new simple and reliable integration method, which is the main contribution of this paper. We propose a new contractor denoted by $\mathcal{C}_{\frac{d}{dt}}$ in association with the differential equation $\dot{x}(t) = f(x(t), t)$.

3.1. CSP approach

An interval integration corresponds to a specific network involving only one constraint given by $\dot{x}(t) = f(x(t), t)$. When intermediate values are known – such as the initial condition $x(0)$ – further constraints can be added to the network. Using a decomposition, our problem reduces to the following CSP:

$$\begin{cases} \textbf{Variables: } x(\cdot),\ v(\cdot) \\ \textbf{Constraints:} \\ \quad 1.\ \dot{x} = v \\ \quad 2.\ v = f(x, t) \\ \textbf{Domains: } [x](\cdot),\ [v](\cdot) \end{cases} \tag{11}$$

The intermediate variable $v(\cdot)$[1] represents the derivative of $x(\cdot)$ (first constraint) and can be computed with an analytical expression (second constraint) or simply known to belong to a given set of trajectories: the tube $[v](\cdot)$.

A reliable resolution is performed by the use of guaranteed tools to implement each constraint. The second constraint, for instance $v = f(x, t) = -\sin(x)$, will be applied with a composition of elementary contractors such as $\mathcal{C}_-$ and $\mathcal{C}_{\sin}$. However, a contractor $\mathcal{C}_{\frac{d}{dt}}$ for the first constraint $\dot{x} = v$ has to be built.

3.2. Differential tube contractor $\mathcal{C}_{\frac{d}{dt}}$

The differential constraint implies contractions that can be propagated along the whole domain in a forward and then a backward way. We break down the implementation into two contractors $\mathcal{C}^{\rightarrow}_{\frac{d}{dt}}$ and $\mathcal{C}^{\leftarrow}_{\frac{d}{dt}}$ so that $\mathcal{C} = \mathcal{C}^{\rightarrow}_{\frac{d}{dt}} \circ \mathcal{C}^{\leftarrow}_{\frac{d}{dt}}$.

Forward contractor $\mathcal{C}^{\rightarrow}_{\frac{d}{dt}}$. When solving an ordinary differential equation numerically such as $\dot{x} = v$, a recurrence relation is typically encountered:

$$x(t + dt) = x(t) + dt \cdot v(t). \tag{12}$$

The new corresponding contractor is obtained with bounded values and intersections:

$$[x](t + dt) = [x](t + dt) \cap ([x](t) + dt \cdot [v](t)). \tag{13}$$

Eventually, this definition has to fit with the chosen representation of a tube built with a set of slices of width δ. This implies some time uncertainties over each slice. Hence, the bounded value of the $(k+1)^{\text{th}}$ slice is obtained thanks to:

$$[x](k+1) = [x](k+1) \cap ([x](k) \cap [x](k+1) + [0, \delta] \cdot [v](k+1)). \tag{14}$$

Backward contractor $\mathcal{C}^{\leftarrow}_{\frac{d}{dt}}$. The same method is applied in backward for the $(k-1)^{\text{th}}$ slice:

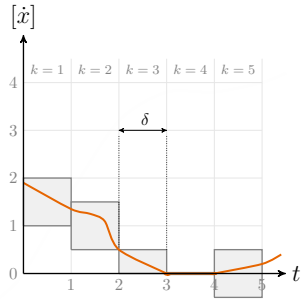
$$[x](k-1) = [x](k-1) \cap ([x](k) \cap [x](k-1) - [0, \delta] \cdot [v](k-1)). \tag{15}$$

The algorithms for tubes contractions are provided below. A simple example of such forward/backward propagation is also given in Figure 5, with a given tube $[\dot{x}](\cdot)$, an uninitialized tube $[x](\cdot)$ and conditions $x_0 = 0$, $x_f = 4$. The variable $[x_{front}]$ used in the algorithms is depicted by blue thick lines in Figure 5.

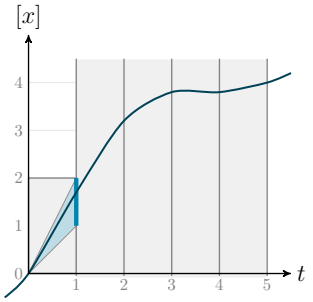
Algorithm 1 $\mathcal{C}^{\rightarrow}_{\frac{d}{dt}}$ (in : $[x_0], [v](\cdot)$, inout : $[x](\cdot)$)

```
1: begin
2: var [x_front] ← [x_0]
3: var [x_old]
4: for k = 1 to k̄ do
5:     [x_old] ← [x](k)
6:     [x](k) ← [x_old] ∩ ([x_front] + [0, δ] · [v](k))
7:     if k ≠ k̄ then
8:         [x_front] ← [x_old] ∩ ([x_front] + δ · [v](k)) ∩ [x](k+1)
9:     end if
10: end for
11: end
```

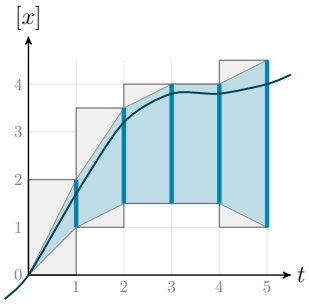
[1] the notation $v(\cdot)$ recalls robot's velocity: the derivative of its position $x(\cdot)$



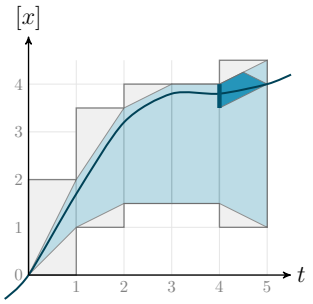
(a) Derivative tube $[\dot{x}](\cdot)$ used to contract $[x](\cdot)$ initialized to $[-\infty,\infty]\forall t$.



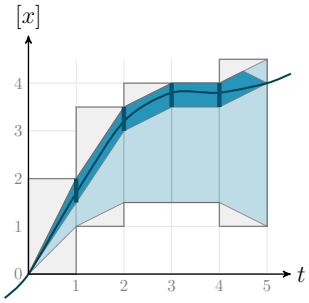
(b) First step of *forward* integration contractor: first slice k_1 is contracted from $[-\infty,\infty]$ to $[0,2]$.



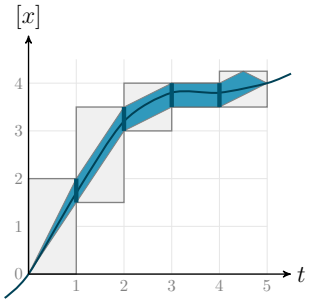
(c) End of *forward* contractions. The minimal envelope of solutions compliant with $[\dot{x}](\cdot)$ is pictured in blue. Slices representation – outer approximation – is depicted in gray.



(d) First step of *backward* integration contractor, with the last condition $x(5) = 4$. A reduced envelope, pictured in dark blue, is computed from $t = 5$ to $t = 0$.



(e) End of *backward* processing.



(f) Tube $[x](\cdot)$ is contracted to optimally enclose the thin envelope pictured in dark blue.

Figure 5: Storyboard of a *forward/backward* tube contraction. Curved lines picture feasible trajectories $\dot{x}^*(\cdot)$ and $x^*(\cdot)$ that remain respectively enclosed within $[\dot{x}](\cdot)$ and $[x](\cdot)$ after the contractions process.

Algorithm 2 $\mathcal{C}_{\frac{d}{dt}}^{\leftarrow}$ (in : $[x_f], [v](\cdot)$, inout : $[x](\cdot)$)

begin
var $[x_{front}] \leftarrow [x_f]$
var $[x_{old}]$
for $k = \overline{k}$ to 1 **do**
  $[x_{old}] \leftarrow [x](k)$
  $[x](k) \leftarrow [x_{old}] \cap ([x_{front}] - [0,\delta] \cdot [v](k))$
  if $k \neq 1$ **then**
    $[x_{front}] \leftarrow [x_{old}] \cap ([x_{front}] - \delta \cdot [v](k)) \cap [x](k-1)$
  end if
end for
end

Algorithm 3 $\mathcal{C}_{\frac{d}{dt}}$ (in : $[x_0], [x_f], [v](\cdot)$, inout : $[x](\cdot)$)

begin
  $\mathcal{C}_{\frac{d}{dt}}^{\rightarrow}([x_0], [v](\cdot), [x](\cdot))$
  $\mathcal{C}_{\frac{d}{dt}}^{\leftarrow}([x_f], [v](\cdot), [x](\cdot))$
end

3.3. Fixed points

A fixed point method may be applied to solve a CSP involving a constraint $v = f(x)$. For instance, the problem $\dot{x} = -\sin(x)$ can be described by a cyclic constraint network as pictured in Figure 6. Due to the existence of one loop, an iterative resolution has to be processed until a fixed point is reached. The proposed approach can be easily applied on this kind of problems but may provide poor results compared with other dedicated libraries such as CAPD [9], which is typically devoted to this type of problem.

However, in most robotics applications, constraints networks form causal kinematic chains. In such cases, our method appears to be efficient and competitive. Section 4 provides mobile robotics examples with a concrete application of the tool.

4. Simulation of a Mobile Robot

4.1. Causal kinematic chain

For mobile robots, the differential equation often has a structure of a causal kinematic chain. For instance, motors generate an acceleration of the wheels rotation, driving the vehicle forward, creating a displacement. As an example,

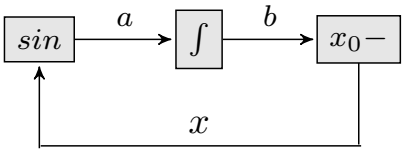


Figure 6: Circuit associated with the IVP defined by equation $\dot{x} = -\sin(x)$. The equation can be broken down into $a = \sin(x)$, $\dot{b} = a$, $x = x_0 - b$. All trajectories $x(\cdot)$, $a(\cdot)$, $b(\cdot)$ belong to tubes.

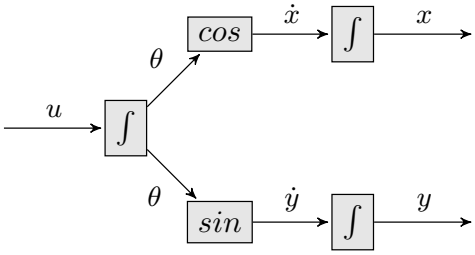


Figure 7: Causal kinematic chains associated to the state equations of a car. All trajectories $u(\cdot)$, $\theta(\cdot)$, $\dot{x}(\cdot)$, $\dot{y}(\cdot)$, $x(\cdot)$, $y(\cdot)$ belong to tubes.

let us consider a wheeled robot $\mathcal{R}$ with a constant velocity [31] and described by the following state equations:

$$\mathcal{R}: \begin{cases} \dot{x}(t) &= 10 \cdot \cos\big(\theta(t)\big), \\ \dot{y}(t) &= 10 \cdot \sin\big(\theta(t)\big), \\ \dot{\theta}(t) &= u(t), \end{cases} \tag{16}$$

where x, y correspond to the position of the robot and θ to its heading. As illustrated by the circuit of Figure 7, the state equations do not consist in looped constraints and are composed of two causal kinematic chains.

Here, car's speed is set to $v = 10m/s$. We assume that the initial state $\mathbf{x}_0$ belongs to the box:

$$\mathbf{x}_0 \in [-1, 1] \times [-1, 1] \times [-6/5\pi - 0.02, -6/5\pi + 0.02].$$

In order to compare with the CAPD library, $u(\cdot)$ is bounded by the following analytical expression:

$$u(t) \in [u](t) = -\cos\left(\frac{t + 33}{5}\right) + [-0.02, 0.02]. \tag{17}$$

Tubes $[\theta](\cdot)$, $[\dot{x}](\cdot)$, $[x](\cdot)$, ..., are initialized to $[-\infty, +\infty]$ for all t. Our propagation method yields the tubes projected on Figure 8. Note that the estimation becomes more pessimistic with time: without exteroceptive measurements, the robot progressively gets lost. The figure shows that our method is more accurate than CAPD on this example.

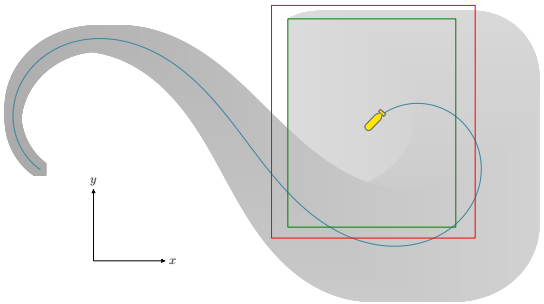


Figure 8: Interval simulation of the robot $\mathcal{R}$. The blue line represents the true poses of the robot while gray shapes correspond to the tubes $[x](\cdot) \times [y](\cdot)$ projected on the world frame. The green box is the final box obtained for $t = 14$. By comparison, the final box computed with CAPD is represented in red.

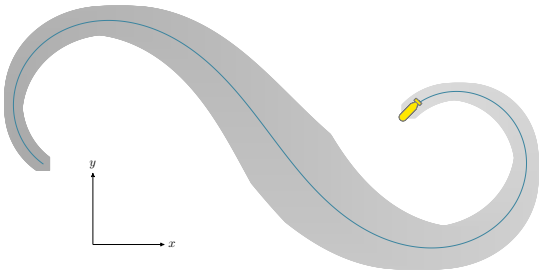


Figure 9: Simulation as presented in Figure 8. This time, the initial and final states are almost known while uncertainties are maximal in the middle of the mission.

To illustrate the fact that our method is more general and more flexible than existing guaranteed integration approaches, consider now a situation where final state $\mathbf{x}(14)$ is known to belong to the box $[\mathbf{x}](14) = [53.9, 55.9] \times [6.9, 8.9] \times [-2.36, -2.32]$. Adding this information to the constraint network, the contractor $\mathcal{C}_{\frac{d}{dt}}^{\leftarrow}$ can be called to perform a backward propagation. The result is illustrated by Figure 9 where backward contractions are observed.

4.2. Example involving higher order differential constraints

Let us now consider a robot described by its state $\mathbf{x} \in \mathbb{R}^2$ representing its position on a horizontal plane, and following a Lissajous trajectory:

$$\mathbf{x}(t) = 5 \cdot \begin{pmatrix} 2\cos(t) \\ \sin(2t) \end{pmatrix}. \tag{18}$$

Equation (18) describes the actual trajectory which is unknown. To illustrate our approach, we generate differential equations satisfied by $\mathbf{x}(t)$. The initial condition is known to belong to a box $[\mathbf{x}_0]$. The associated constraint network is the following:

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot),\ \dot{\mathbf{x}}(\cdot),\ \ddot{\mathbf{x}}(\cdot) \\ \textbf{Constraints:} \\ \quad 1.\ \ddot{x}_1(t) \in -10\cos(t) + [-0.001, 0.001] \\ \quad 2.\ \ddot{x}_2 = -0.4 \cdot \dot{x}_1 \cdot \ddot{x}_1 \\ \quad 3.\ \dot{\mathbf{x}}(0) = \begin{pmatrix} 0 \\ 10 \end{pmatrix},\ \mathbf{x}(0) \in \begin{pmatrix} [9.8, 10.2] \\ [-0.2, 0.2] \end{pmatrix} \\ \textbf{Domains: } [\mathbf{x}](\cdot),\ [\dot{\mathbf{x}}](\cdot),\ [\ddot{\mathbf{x}}](\cdot) \end{cases} \tag{19}$$

The contractor based approach provides the envelope of trajectories pictured in Figure 10. It shows the ease-of-use of differential constraints that are usually encountered in mobile robotics where $\mathbf{x}$ is the position of a robot, $\dot{\mathbf{x}}$ its speed and $\ddot{\mathbf{x}}$ its acceleration.

5. Extension to State Estimation

A state estimation problem (see *e.g.* [32, 33, 34]) involving state constraints can be cast into a constraint

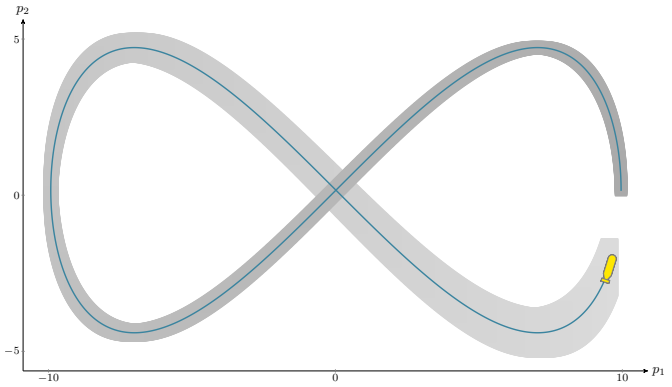


Figure 10: A robot following a Lissajous curve. Blue line is the truth given by Equation (18). Gray area is the envelope of trajectories computed with the proposed method applied on the previous CSP. One should note that further constraints such as $x_2(t) = x_2(t+\pi)$ or $\mathbf{x}(\frac{\pi}{2}) = \mathbf{x}(\frac{3\pi}{2})$ could be easily added to the CSP.

network for which four types of uncertainty propagations are encountered. The *forward* propagation (1), the *backward* propagation (2), the *correction* (3) and the *state constraints* (4). In the literature, (1) is known as the *prediction*, (1)+(2) the *integration*, (3) the *correction*, (1)+(3) the *filter* and (1)+(2)+(3) the *smoother*. Section 3 provides a tool for (1)+(2), *i.e.*, the forward/backward integration. This has been illustrated in Section 4 with the simulation of mobile robots. In the current section, we will show that the approach can also be extended to all types of constraints, namely (1)+(2)+(3)+(4), which is of high interest for robotics applications. An example is given on a real dataset.

5.1. Added corrections and state constraints

A state estimation relies on the following equations:

$$\begin{cases} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), t), \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t)), \end{cases} \tag{20}$$

where $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is the *evolution* function and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ is the *observation* function. The latter allows corrections over the state $\mathbf{x}(t)$ from a given measurement $\mathbf{y}(t)$ that can be bounded by $[\mathbf{y}](t)$. This constraint is easily applied by performing a local contraction of the tube $[\mathbf{x}](\cdot)$ at time t such that:

$$[\mathbf{x}](t) = [\mathbf{x}](t) \cap \mathbf{g}^{-1}([\mathbf{y}](t)). \tag{21}$$

A *filter* or a *smoother* procedure is then respectively obtained thanks to $\mathcal{C}_{\frac{d}{dt}}^{\rightarrow}$ or $\mathcal{C}_{\frac{d}{dt}}$.

5.2. State estimation of an underwater robot

As an example, let us consider a real experiment with an underwater robot. We will consider both evolution equations and state observations in order to perform a complete state estimation. A classical kinematic model for an underwater robot [35, 2] is:

$$\begin{cases} \dot{\mathbf{p}} &= \mathbf{R}(\psi, \theta, \varphi) \cdot \mathbf{v}_r, \\ \dot{\mathbf{v}}_r &= \mathbf{a}_r - \boldsymbol{\omega}_r \wedge \mathbf{v}_r, \end{cases} \tag{22}$$

where $\mathbf{R}(\psi, \theta, \varphi)$ is the Euler matrix given by:

$$\begin{pmatrix} \cos\theta\cos\psi & -\cos\varphi\sin\psi + \sin\theta\cos\psi\sin\varphi & \sin\psi\sin\varphi + \sin\theta\cos\psi\cos\varphi \\ \cos\theta\sin\psi & \cos\psi\cos\varphi + \sin\theta\sin\psi\sin\varphi & -\cos\psi\sin\varphi + \sin\theta\cos\varphi\sin\psi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{pmatrix}.$$

In these equations, the vector $\mathbf{p} = (p_x, p_y, p_z)^\intercal$ gives the coordinates of the center of the robot expressed in the absolute inertial coordinate system $\mathcal{R}_0$, as well as the three Euler angles (ψ, θ, φ). The robot's speed vector $\mathbf{v}_r$ and acceleration vector $\mathbf{a}_r$ are expressed in its own coordinate system $\mathcal{R}_1$. The vector $\boldsymbol{\omega}_r = (\omega_x, \omega_y, \omega_z)^\intercal$ corresponds to the rotation vector of the robot relative to $\mathcal{R}_0$ expressed in $\mathcal{R}_1$. It is indeed conventional to express $\mathbf{a}_r$, $\boldsymbol{\omega}_r$ in the coordinate system of the robot since these quantities are generally measured by the robot itself *via* an inertial measurement unit attached on it.

Consider now a trajectory of the Autonomous Underwater Vehicle DAURADE (Figure 11). From its inertial unit, we collect measurements for $\mathbf{a}_r$, $\boldsymbol{\omega}_r$, ψ, θ, φ with some bounded errors defined thanks to the datasheet of the sensor. Furthermore, DAURADE is equipped with a DVL[2] giving measurements about its speed vector $\mathbf{v}_r$. From these sensors we create tubes based on each of these variables. See for instance Figure 13.

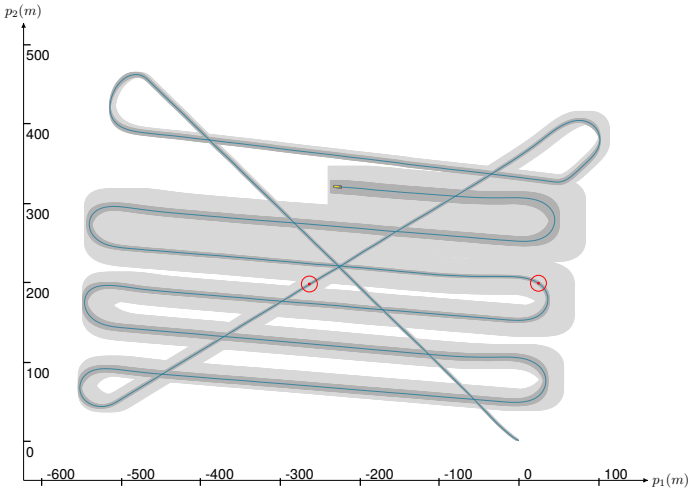
Figure 11: DAURADE AUV managed by *DGA Techniques Navales Brest* and the *Service Hydrographique et Océanographique de la Marine* (SHOM), during an experiment in the Rade de Brest, October 2015. *Photo: S. Rohou.*

With tube arithmetic and tube integration, we finally compute the positions of the robot: $[\mathbf{p}](\cdot)$. One of its component is plotted in Figure 14. An overview of the mission is pictured by Figure 12 where the first dead reckoning computation is depicted in light gray. A state estimation is then performed with the help of two positioning measurements pictured in red. This led to the contraction of $[\mathbf{p}](\cdot)$ over the whole mission duration, by using the presented contractor $\mathcal{C}_{\frac{d}{dt}}$.

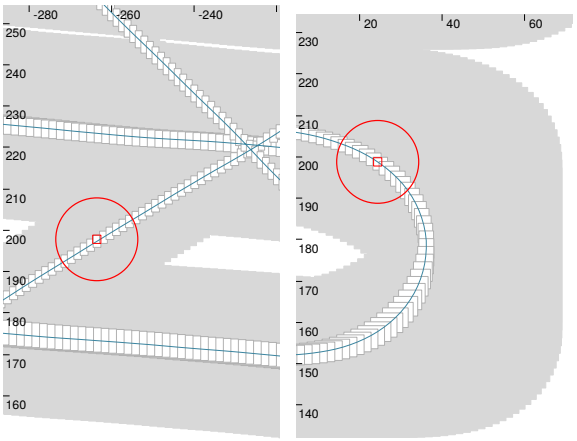
6. Conclusions

This paper proposes a simple and efficient constraint-based method for the guaranteed interval computation of

[2]DVL: Doppler Velocity Log, a hydro-acoustic sensor using the Doppler effect to measure robot's velocity over the seabed.



(a) Mission map: DAURADE exploring a 25 hectares seabed area.



(b) Zoom on the two state observations: these are bounded positioning measurements obtained at $t_1 = 14\,\text{min}$ and $t_2 = 32\,\text{min}$.

Figure 12: Mission map after 45 minutes. Blue line is DAURADE's true trajectory, given by an ultra-short baseline (USBL), a system made of underwater acoustic sensors for positioning purposes. Gray area and white boxes respectively correspond to the tubes $[p_1](\cdot) \times [p_2](\cdot)$ projected on the world frame before and after the consideration of fleeting positioning measurements, pictured by red boxes. Indeed, the only use of inertial data and velocity measurements provided by the DVL leads to a strong drift at the end of the mission. Here, thanks to two observations, the drift is reduced over the whole mission duration.

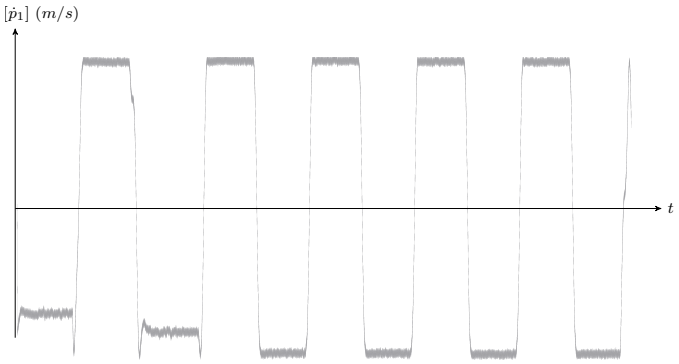


Figure 13: Example of DAURADE's tube $[\dot{p}_1](\cdot)$ computed to represent the velocity along x in $\mathcal{R}_0$. Because this tube comes from measurements without any integration process, its thickness remains constant.

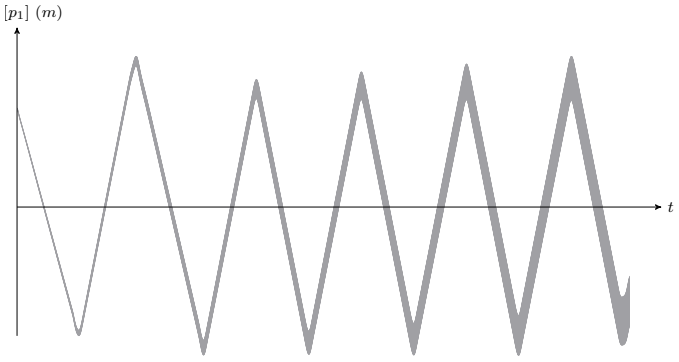


Figure 14: DAURADE's tube $[p_1](\cdot)$ computed as the primitive of $[\dot{p}_1](\cdot)$. This tube becomes thicker depicting cumulative uncertainties due to the dead-reckoning method. Observations pictured in Figure 12 are not represented here.

mobile robots trajectories. The principle is to model the problem as a constraint network and generate a contractor from each constraint. The contractors contract all tubes as much as possible up to a fixed point. In order to deal with dynamical systems, a dedicated differential contractor has been proposed. Its use is well suited in high order differential contexts, non-linear equations, fleeting observations and real datasets. Hence, this approach appears to be competitive for robotics applications while providing reliable sets of robots trajectories.

7. Available Libraries

An optimized tube class has been implemented during this work and is available on: www.simon-rohou.fr/research/tubint. The source code of the simulated examples presented in this paper are also provided in this library. This class is compatible with IBEX: a C++ library for system solving and global optimization based on interval arithmetic and constraint programming, see www.ibex-lib.org. Figures have been drawn using the visualizer VIBEs [36].

8. Acknowledgments

This work has been supported by the French *Direction Générale de l'Armement* (DGA) during the UK-France PhD program. We also thank *DGA Techniques Navales Brest* for their kind help during DAURADE's experiments.

[1] M. Konecny, W. Taha, J. Duracz, A. Duracz, A. Ames, Enclosing the behavior of a hybrid system up to and beyond a zeno point, in: Cyber-Physical Systems, Networks, and Applications

(CPSNA), 2013 IEEE 1st International Conference on, IEEE, 2013, pp. 120–125.
[2] L. Jaulin, Mobile Robotics, Elsevier Science, 2015.
[3] W. Taha, A. Duracz, Acumen: An Open-Source Testbed for Cyber-Physical Systems Research, Springer International Publishing, Cham, 2016, pp. 118–130.
[4] R. E. Moore, Methods and applications of interval analysis, SIAM, 1979.
[5] M. Berz, Computational differentiation: techniques, applications, and tools, no. 89, Society for Industrial & Applied, 1996.
[6] N. Revol, K. Makino, M. Berz, Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY, The Journal of Logic and Algebraic Programming 64 (1) (2005) 135–154.
[7] N. S. Nedialkov, K. R. Jackson, G. F. Corliss, Validated solutions of initial value problems for ordinary differential equations, Applied Mathematics and Computation 105 (1) (1999) 21–68.
[8] J. A. dit Sandretto, A. Chapoutot, Validated explicit and implicit runge-kutta methods, Reliable Computing 22 (2016) 79.
[9] D. Wilczak, P. Zgliczyński, et al., Cr-lohner algorithm, Schedae Informaticae 2011 (Volume 20) (2012) 9–42.
[10] N. Ramdani, N. S. Nedialkov, Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques, Nonlinear Analysis: Hybrid Systems 5 (2) (2011) 149–162.
[11] P. Collins, A. Goldsztejn, The reach-and-evolve algorithm for reachability analysis of nonlinear dynamical systems, Electronic Notes in Theoretical Computer Science 223 (2008) 87 – 102.
[12] E. Goubault, O. Mullier, S. Putot, M. Kieffer, Inner approximated reachability analysis, in: Proceedings of the 17th international conference on Hybrid systems: computation and control, ACM, 2014, pp. 163–172.
[13] L. Jaulin, Nonlinear bounded-error state estimation of continuous-time systems, Automatica 38 (6) (2002) 1079–1082.
[14] W. Tucker, The lorenz attractor exists, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 328 (12) (1999) 1197–1202.
[15] A. Goldsztejn, W. Hayes, P. Collins, Tinkerbell is chaotic, SIAM Journal on Applied Dynamical Systems 10 (4) (2011) 1480–1501.
[16] P. Van Hentenryck, L. Michel, Y. Deville, Numerica: a modeling language for global optimization, MIT press, 1997.
[17] C. Bessiere, Constraint propagation, Foundations of Artificial Intelligence 2 (2006) 29–83.
[18] A. K. Mackworth, Consistency in networks of relations, Artificial intelligence 8 (1) (1977) 99–118.
[19] I. Araya, G. Trombettoni, B. Neveu, A contractor based on convex interval Taylor, in: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Springer, 2012, pp. 1–16.
[20] F. Le Bars, J. Sliwka, L. Jaulin, O. Reynet, Set-membership state estimation with fleeting data, Automatica 48 (2) (2012) 381–387.
[21] A. Bethencourt, L. Jaulin, Solving non-linear constraint satisfaction problems involving time-dependant functions, Mathematics in Computer Science 8 (3) (2014) 503–523.
[22] L. Jaulin, Applied interval analysis: with examples in parameter and state estimation, robust control and robotics, Vol. 1, Springer Science & Business Media, 2001.
[23] C. Combastel, A state bounding observer for uncertain nonlinear continuous-time systems based on zonotopes, in: Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on, IEEE, 2005, pp. 7228–7234.
[24] S. Veres, Error control in polytope computations, Journal of Optimization Theory and Applications 113 (2) (2002) 325–355.
[25] V. Drevelle, P. Bonnifait, Localization confidence domains via set inversion on short-term trajectory, IEEE Transactions on Robotics 29 (5) (2013) 1244–1256.
[26] G. Chabert, L. Jaulin, Contractor programming, Artificial Intelligence 173 (11) (2009) 1079–1100.
[27] A. Kurzhanski, I. Valyi, Ellipsoidal calculus for estimation and control, Nelson Thornes, 1997.
[28] M. Milanese, J. Norton, H. Piet-Lahanier, É. Walter, Bounding approaches to system identification, Springer Science & Business Media, 2013.
[29] C. Aubry, R. Desmare, L. Jaulin, Loop detection of mobile robots using interval analysis, Automatica 49 (2) (2013) 463 – 470.
[30] M. Nehmeier, J. W. von Gudenberg, filib++ , expression templates and the coming interval standard, Reliable Computing 15 (2011) 312–320.
[31] L. E. Dubins, On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, American Journal of mathematics 79 (3) (1957) 497–516.
[32] J. Norton, S. M. Veres, Outliers in bound-based state estimation and identification, in: Circuits and Systems, 1993., ISCAS'93, 1993 IEEE International Symposium on, IEEE, 1993, pp. 790–793.
[33] D. Bertsekas, I. Rhodes, Recursive state estimation for a set-membership description of uncertainty, IEEE Transactions on Automatic Control 16 (2) (1971) 117–128.
[34] A. Gning, S. Julier, L. Mihaylova, Non-linear state estimation using imprecise samples, in: Proceedings of the 16th International Conference on Information Fusion, IEEE, 2013, pp. 2110–2116.
[35] T. I. Fossen, Guidance and control of ocean vehicles, John Wiley & Sons Inc, 1994.
[36] V. Drevelle, J. Nicola, Vibes: A visualizer for intervals and boxes, Mathematics in Computer Science 8 (3-4) (2014) 563–572.