Formal Verification of Robotic Behaviors in Presence of Bounded Uncertainties

Julien Alexandre Dit Sandretto, Alexandre Chapoutot, Olivier Mullier

Formal Verification of Robotic Behaviors in Presence of Bounded Uncertainties

Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier
ENSTA ParisTech, Université Paris-Saclay,
828, bd des maréchaux, 91762 Palaiseau Cedex, France
Email: {alexandre, chapoutot, mullier}@ensta-paristech.fr

March 29, 2017

Abstract

Robotic behaviors are mainly described by differential equations. Those mathematical models are usually not precise enough because of inaccurately known parameters or model simplifications. Nevertheless, robots are often used in critical contexts as medical or military fields. So, uncertainties in mathematical models have to be taken into account in order to produce reliable and safe analysis results. A framework based on interval analysis is proposed to safely verify and analyze robotic behaviors with bounded uncertainties. It follows an interval constraint programming approach, combined with validated numerical integration methods to deal with differential equations. A case study on robust path planning is presented to emphasize the efficiency of the complete framework.

1 Introduction

Designing control algorithms of robots is a challenging activity in order to guarantee the success of the mission and taking into account safety constraints. Moreover, control architecture is made of several layers strongly interacting with each other which makes the design complex. Indeed, several kinds of algorithms ranging from *Proportional-Integral-Derivative* controllers (PID controllers) to neural networks are used and combined to achieve the missions of robots.

In control theory, mathematical models of the controlled element are essential to define control algorithms. Nevertheless, such models are usually a simplification of the real world. Mainly, accurate models are expensive to define, or even impossible, due to the presence of unknown, or hardly measurable, values of the involved parameters (*e.g.*, the inertial moment). Mathematical models represent possible behaviors of robots in function of the control algorithms. So they play a critical role in the design or in the verification process of robots.

The framework given by *constraint programming* [1] is powerful enough to solve many problems with numerous constraints. Unfortunately, taking into account the dynamic of robots is not easy in this framework. In contrary, *validated numerical integration methods* are mature techniques to simulate, with bounded uncertainties, dynamical systems described by differential equations [2, 3, 4, 5]. While such methods produce rigorous enclosures of the trajectories, they can not consider constraints on the dynamics without major modifications. Our goal is to bridge the gap between these two approaches.

In this article, we propose an approach to analyze or verify control algorithms of robots based on mathematical models of their behaviors. An extension of constraint programming framework with differential equations is proposed, hence our models can take into account bounded uncertainties on the parameters offering guarantees on robot behaviors, while taking into account temporal constraints as safety. We extend current work on this field, such as [6, 7, 8], by considering more complex constraints involving time.

Formal verification of robotic application is not new and several previous work address this problem. Nevertheless, they usually rely on temporal logic specification as *Linear Temporal Logic* (LTL) to prove the correctness. For example, this approach is used in [9, 10]. *Satisfiability Modulo Theories* techniques (SMT techniques), which are described in [11, 12], can also be used to prove safety or even stability of hybrid systems. Our contribution differs from these works by adopting constraint satisfaction problem formulation in order to easily specify the requirements.

The paper is organized as follows. In Section 2, the basics of interval analysis, constraint satisfaction problem and guaranteed numerical integration are provided. The main contribution is presented in sections 3 and 4. A case study is described in Section 5 before concluding in Section 6.

2 Preliminary Notions

A presentation of the main mathematical tools is given in this section. First, basics of *interval analysis* is provided in Section 2.1. Second, Section 2.2 gives a quick presentation of *constraint satisfaction problem*. Finally, a short introduction of *validated numerical integration* is presented in Section 2.3.

2.1 Interval Analysis

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* (see [13]). An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathcal{IR}$ denotes the set of all intervals over reals. The size or the width of $[x_i]$ is denoted by $w([x_i]) = \overline{x_i} - \underline{x_i}$.

Interval arithmetic extends to $\mathcal{IR}$ elementary functions over $\mathcal{R}$. For instance, the interval sum, *i.e.*, $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$, encloses the image of the sum function over its arguments.

An interval vector or a *box* $[\mathbf{x}] \in \mathcal{IR}^n$, is a Cartesian product of n intervals. The enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

Definition 1 (Inclusion function) *Consider a function $f : \mathcal{R}^n \to \mathcal{R}^m$, then $[f] : \mathcal{IR}^n \to \mathcal{IR}^m$ is said to be an extension of f to intervals if*

$$\forall [\mathbf{x}] \in \mathcal{IR}^n, \quad [f]([\mathbf{x}]) \supseteq \{f(\mathbf{x}), \mathbf{x} \in [\mathbf{x}]\} \ .$$

It is possible to define inclusion functions for all elementary functions such as $\times$, $\div$, sin, cos, exp, and so on. The *natural* inclusion function is the simplest to obtain: all occurrences of the real variables are replaced by their interval counterpart and all arithmetic operations are evaluated using interval arithmetic. More sophisticated inclusion functions such as the centered form, or the Taylor inclusion function may also be used (see [14] for more details).

Example 1 (Interval arithmetic) *A few examples of arithmetic operations between interval values are given*

$$\begin{aligned}
[-2,5] + [-8,12] &= [-10,17] \\
[-10,17] - [-8,12] = [-10,17] + [-12,8] &= [-22,25] \\
[-10,17] - [-2,5] &= [-15,19] \\
\frac{[-2,5]}{[-8,12]} &= [-\infty,\infty] \\
\frac{[3,5]}{[8,12]} &= \left[\frac{3}{12}, \frac{5}{8}\right] \\
\left[\frac{3}{12}, \frac{5}{8}\right] \times [8,12] &= \left[2, \frac{15}{2}\right]
\end{aligned}$$

In the first example of division, the result is the interval containing all the real numbers because denominator contains 0.

As an example of inclusion function, we consider a function p defined by

$$p(x, y) = xy + x \ .$$

The associated natural inclusion function is

$$[p]([x], [y]) = [x][y] + [x],$$

in which variables, constants and arithmetic operations have been replaced by its interval counterpart. And so
$p([0,1],[0,1]) = [0,2] \subseteq \{p(x,y) \mid x, y \in [0,1]\} = [0,2]$. ■

2.2 Numerical Constraint Satisfaction Problems

In this section, we introduce the numerical constraint satisfaction problem (NCSP) formalism, following the description given in [1], and present some basics on constraint programming.

A NCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{V} := \{v_1, \ldots, v_n\}$ is a finite set of variables which can also be represented by the vector $\mathbf{v}$;

- $\mathcal{D} := \{[v_1], \ldots, [v_n]\}$ is a set of intervals such that $[v_i]$ contains all possible values of v_i. It can be represented by a box $[\mathbf{v}]$ gathering all $[v_i]$;
- $\mathcal{C} := \{c_1, \ldots, c_m\}$ is a set of constraints of the form $c_i(\mathbf{v}) \equiv f_i(\mathbf{v}) = 0$ or $c_i(\mathbf{v}) \equiv g_i(\mathbf{v}) \leqslant 0$, with $f_i : \mathcal{R}^n \to \mathcal{R}$, $g_i : \mathcal{R}^n \to \mathcal{R}$ for $1 \leqslant i \leqslant m$. Constraints $\mathcal{C}$ are interpreted as a conjunction of equalities and inequalities.

The solution of a NCSP is a valuation of $\mathbf{v}$ ranging in $[\mathbf{v}]$ and satisfying the constraints $\mathcal{C}$.

The approach of NSCP is both powerful to address complex problems (NP-hard problem with numerical issues, even in critical applications) and simple in the definition of a solving framework [15, 16]. Indeed, the classical algorithm to solve a NCSP is the branch-and-prune which needs only an evaluation of the constraints and an initial domain for variables. If this algorithm is sufficient for many problems, some improvements have been achieved and the algorithms based on contractors have emerged [17]. The branch-and-contract algorithm consists in two main steps i) the contraction (or filtering) of one variable and the propagation to the others till a fixed point is reached, then ii) the bisection of the domain of one variable in order to obtain two problems, easier to solve. A more detailed description follows.

2.2.1 Contraction

A filtering algorithm or contractor is used in a NCSP solver to reduce the domain of variables till a fixed point (or close to), by respecting the local consistencies. A contractor Ctc can be defined with the help of constraint programming, analysis or algebra, but it must satisfy three properties:

- $Ctc(\mathcal{D}) \subseteq \mathcal{D}$: the contractance,
- Ctc cannot remove any solution: it is conservative,
- $\mathcal{D}' \subseteq \mathcal{D} \Rightarrow Ctc(\mathcal{D}') \subseteq Ctc(\mathcal{D})$: the monotonicity.

There are many contractor operators defined in the literature, among them the most notable are:

- (Forward-Backward contractor) By considering only one constraint, this method computes the interval enclosure of a node in the tree of constraint operations with the children domains (the forward evaluation), then refines the enclosure of a node in terms of parents domain (the backward propagation). For example, from the constraint $x + y = z$, this contractor refines initial domains $[x]$, $[y]$ and $[z]$ from a forward evaluation $[z] = [z] \cap ([x] + [y])$, and from two backward evaluations $[x] = [x] \cap ([z] - [y])$ and $[y] = [y] \cap ([z] - [x])$.
- (Newton contractor) This contractor, based on first order Taylor interval extension: $[f]([\mathbf{x}]) = f(\mathbf{x}^*) + [J_f]([\mathbf{x}])([\mathbf{x}] - \mathbf{x}^*)$ with $\mathbf{x}^* \in [\mathbf{x}]$, has the property of
  - if $0 \in [f]([\mathbf{x}])$,
  - then $[\mathbf{x}]_{k+1} = [\mathbf{x}]_k \cap x^* - [J_f]([\mathbf{x}]_k)^{-1} f(\mathbf{x}^*)$ is a tighter inclusion of the solution of $f(x) = 0$.

  Some other contractors based on Newton have been defined such as Krawczyk [14].

2.2.2 Propagation

If a variable domain has been reduced, the reduction is propagated to all the constraints involving that variable, which allows to narrow the other variable domains. This process is repeated till a fixed point is reached.

2.2.3 Branch-and-Prune

A Branch-and-Prune algorithm consists on alternatively branching and pruning to produce two sub-pavings $\mathcal{L}$ and $\mathcal{S}$, with $\mathcal{L}$ the boxes too small to be bisected and $\mathcal{S}$ the solution boxes. We are then sure that all solutions are included in $\mathcal{L} \cup \mathcal{S}$ and that every point in $\mathcal{S}$ is solution.

Literally, this algorithm browses a list of boxes $\mathcal{W}$, initially $\mathcal{W}$ is set with a vector $[\mathbf{x}]$ made of the elements of $\mathcal{D}$, and for each one i) Prune: the NCSP is evaluated (or contracted) on the current box, if it is a solution it is added to $\mathcal{S}$, otherwise ii) Branch: if the box is large enough, it is bisected and the two boxes resulting are added into $\mathcal{W}$, otherwise the box is added to $\mathcal{L}$.

Example 2 *An example of the problems that the previously presented tools can solve is taken from [18]. The considered NCSP is defined such that*

- $\mathcal{V} = \{x, y, z, t\}$
- $\mathcal{D} = \{[x] = [0, 1000], [y] = [0, 1000],$
  $[z] = [0, 3.1416], [t] = [0, 3.1416]\}$
- $\mathcal{C} = \{xy + t - 2z = 4; x\sin(z) + y\cos(t) = 0; x - y + \cos^2(z) = \sin^2(t); xyz = 2t\}$

To solve this CSP we use a Branch-and-Prune algorithm with the Forward-Backward contractor and a propagation algorithm. We obtain one solution ($[1.999, 2.001]$, $[1.999, 2.001]$, $[1.57, 1.571]$, $[3.14159, 3.1416]$*). The algorithm needs* 6 *bisections.* ■

2.3 Validated Numerical Integration Methods

A differential equation is a mathematical relation used to define an unknown function by its derivatives. These derivatives usually represent the temporal evolution of a physical quantity. Because the majority of systems is defined by the evolution of a given quantity, differential equations are used in engineering, physics, chemistry, biology or economics as examples. Mathematically, differential equations have no explicit solutions, except for few particular cases. Nevertheless, the solution can be numerically approximated with the help of integration schemes such as Taylor series [2] or Runge-Kutta methods [5, 4].

In the following, we consider a *generic* parametric differential equation as an *interval initial value problem* (IIVP) defined by

$$\begin{cases} \dot{\mathbf{y}} = F(t, \mathbf{y}, \mathbf{x}, \mathbf{p}, \mathbf{u}) \\ 0 = G(t, \mathbf{y}, \mathbf{x}, \mathbf{p}, \mathbf{u}) \\ \mathbf{y}(0) \in \mathcal{Y}_0, \mathbf{x}(0) \in \mathcal{X}_0, \mathbf{p} \in \mathcal{P}, \mathbf{u} \in \mathcal{U}, t \in [0, t_{\text{end}}] \ , \end{cases} \tag{1}$$

with $F : \mathcal{R} \times \mathcal{R}^n \times \mathcal{R}^m \times \mathcal{R}^r \times \mathcal{R}^s \mapsto \mathcal{R}^n$ and $G : \mathcal{R} \times \mathcal{R}^n \times \mathcal{R}^m \times \mathcal{R}^r \times \mathcal{R}^s \mapsto \mathcal{R}^m$. The variable $\mathbf{y}$ of dimension n is the differential variable while the variable $\mathbf{x}$ is an algebraic variable of dimension m with an initial condition $\mathbf{y}(0) \in \mathcal{Y}_0 \subseteq \mathcal{R}^n$ and $\mathbf{x}(0) \in \mathcal{X}_0 \subseteq \mathcal{R}^n$. In other words, differential-algebraic equations (DAE) of index 1 are considered, and in the case of $m = 0$, this differential equation simplifies to an ordinary differential equation (ODE). Note that usually, the initial values of algebraic variable $\mathbf{x}$ are computed by numerical algorithms used to solve DAE but we consider it fixed here for simplicity. Variable $\mathbf{p} \in \mathcal{P} \subseteq \mathcal{R}^r$ stands for parameters of dimension r and variable $\mathbf{u} \in \mathcal{U} \subseteq \mathcal{R}^s$ stands for a control vector of dimension s. We assume standard hypotheses on F and G to guarantee the existence and uniqueness of the solution of such problem.

A validated simulation of a differential equation consists in a discretization of time, such that $t_0 \leqslant \cdots \leqslant t_{\text{end}}$, and a computation of enclosures of the set of states of the system $\mathbf{y}_0$, ..., $\mathbf{y}_{\text{end}}$, by the help of a guaranteed integration scheme. In details, a guaranteed integration scheme is made of

- an integration method $\Phi(F, G, \mathbf{y}_j, t_j, h)$, starting from an initial value $\mathbf{y}_j$ at time t_j and a finite time horizon h (the step-size), producing an approximation $\mathbf{y}_{j+1}$ at time $t_{j+1} = t_j + h$, of the exact solution $\mathbf{y}(t_{j+1}; \mathbf{y}_j)$, *i.e.*, $\mathbf{y}(t_{j+1}; \mathbf{y}_j) \approx \Phi(F, G, \mathbf{y}_j, t_j, h)$;
- a truncation error function $\text{lte}_\Phi(F, G, \mathbf{y}_j, t_j, h)$, such that $\mathbf{y}(t_{j+1}; \mathbf{y}_j) = \Phi(F, G, \mathbf{y}_j, t_j, h) + \text{lte}_\Phi(F, G, \mathbf{y}_j, t_j, h)$.

Basically, a validated numerical integration method is based on a numerical integration scheme such as Taylor series [2] or Runge-Kutta methods [5, 4] which is extended with interval analysis tools to bound the *truncation error*, *i.e.*, the distance between the exact and the numerical solutions. Mainly, such methods work in two stages at each integration step, starting from an enclosure $[\mathbf{y}_j] \ni \mathbf{y}(t_j; \mathbf{y}_0)$ at time t_j of the exact solution, we proceed by:

i. a computation of an *a priori* enclosure $[\widetilde{\mathbf{y}}_{j+1}]$ of the solution $\mathbf{y}(t; \mathbf{y}_0)$ for all t in the time interval $[t_j, t_{j+1}]$. This stage allows one to prove the existence and the uniqueness of the solution.

ii. a computation of a tightening of state variable $[\mathbf{y}_{j+1}] \ni \mathbf{y}(t_{j+1}; \mathbf{y}_0)$ at time t_{j+1} using $[\widetilde{\mathbf{y}}_{j+1}]$ to bound the term $\text{lte}_\Phi(F, G, \mathbf{y}_j, t_j, h)$.

Sometimes, additive constraints can be considered, coming from a mechanical constraint, an energy conservation, a control law, or whatever [5]. It is important to understand that these constraints are linked to the system, valid all the time, and thus semantically different from a constraint coming from a satisfaction problem which are valid only at particular time instants. Finally a *constrained parametric IIVP* is considered

$$\begin{cases} \dot{\mathbf{y}} = F(t, \mathbf{y}, \mathbf{x}, \mathbf{p}, \mathbf{u}) \\ 0 = G(t, \mathbf{y}, \mathbf{x}, \mathbf{p}, \mathbf{u}) \\ 0 = H(t, \mathbf{y}, \mathbf{p}, \mathbf{u}) \\ \mathbf{y}(0) \in \mathcal{Y}_0, \mathbf{x}(0) \in \mathcal{X}_0, \mathbf{p} \in \mathcal{P}, \mathbf{u} \in \mathcal{U}, t \in [0, t_{\text{end}}] \ , \end{cases} \tag{2}$$

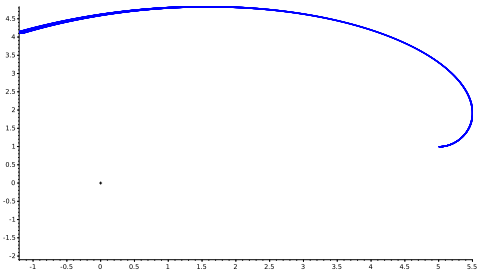

Figure 1: Results of validated simulation: y_2 w.r.t. y_1.

where $H : \mathcal{R} \times \mathcal{R}^n \times \mathcal{R}^r \times \mathcal{R}^s \mapsto \mathcal{R}^c$.

Methods defined in [5, 4] can take into account such kind of constraints by adapting the first stage of validated numerical integration methods, that is the computation of the *a priori* enclosure.

A validated simulation starts with the interval enclosures $[\mathbf{y}(0)]$, $[\mathbf{x}(0)]$, $[\mathbf{p}]$ and $[\mathbf{u}]$ of respectively, $\mathcal{Y}_0$, $\mathcal{X}_0$, $\mathcal{P}$, and $\mathcal{U}$. It produces two lists of boxes:

- the list of discretization time steps. $\{[\mathbf{y}_0], \ldots, [\mathbf{y}_{\text{end}}]\}$;
- the list of *a priori* enclosures: $\{[\widetilde{\mathbf{y}}_0], \ldots, [\widetilde{\mathbf{y}}_{\text{end}}]\}$.

Based on these lists, two functions depending on time can be defined

$$R : \left\{ \begin{array}{c} \mathcal{R} \mapsto \mathcal{IR}^n \\ t \to [\mathbf{y}] \end{array} \right. \tag{3}$$

with $\{\mathbf{y}(t; \mathbf{y}_0) : \forall \mathbf{y}_0 \subseteq [\mathbf{y}_0]\} \subseteq [\mathbf{y}]$, and

$$\widetilde{R} : \left\{ \begin{array}{c} \mathcal{IR} \mapsto \mathcal{IR}^n \\ [\underline{t}, \overline{t}] \to [\widetilde{\mathbf{y}}] \end{array} \right. \tag{4}$$

with $\{\mathbf{y}(t; \mathbf{y}_0) : \forall \mathbf{y}_0 \in [\mathbf{y}_0] \wedge \forall t \in [\underline{t}, \overline{t}]\} \subseteq [\widetilde{\mathbf{y}}]$.

Function R, defined in (3), is obtained by new applications of validated integration method starting from $[\mathbf{y}_k]$ at t_k, and finishing at t with $t_k < t < t_{k+1}$. Function $\widetilde{R}$, defined in (4), is obtained with the union of $[\widetilde{\mathbf{y}}_k]$ with $k = a, \ldots, b$ and $t_a < \underline{t} < \overline{t} < t_b$. These functions are then strictly conservative.

More abstractly, the functions R and $\widetilde{R}$ define two interval enclosures of the solution function of differential equations defined in Equation (2).

Example 3 *We consider an Initial Value Problem based on the following constrained differential algebraic equation*

$$\left\{ \begin{array}{l} F : \left\{ \begin{array}{l} \dot{y_1} = -y_3 y_2 - (1 + y_3) x_1 \\ \dot{y_2} = y_3 y_1 - (1 + y_3) x_2 \\ \dot{y_3} = 1 \end{array} \right. \\ G : \left\{ \begin{array}{l} 0 = (y_1 - x_2)/5 - \cos(y_3^2/2) \\ 0 = (y_2 - x_1)/5 - \sin(y_3^2/2) \end{array} \right. \\ H : \{ \, 0 = x_1^2 + x_2^2 - 1 \end{array} \right. \tag{5}$$

and the initial values $\mathbf{y}(0) = (5, 1, 0)$ *and* $\mathbf{x}(0) = (-1, 0)$. *Interesting variables are* y_1 *and* y_2, *the states of the system. A picture of the result of validated simulation is given in Figure 1 and Figure 2.*

It follows an example of the use of the functions providing enclosures at given time $R(t)$ *and* $\widetilde{R}([t])$*:*

- $R(1.1) = ([5.001, 5.006]; [3.294, 3.299]; [1.099, 1.100])$
- $\widetilde{R}([0.7, 0.8]) =$
  $([5.463, 5.495]$*;* $[1.975, 2.270]$*;* $[0.699, 0.800])$

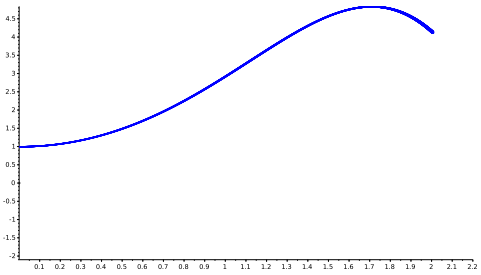

Figure 2: Results of validated simulation: y_2 w.r.t. time.

The problem given by Equation (5) *has a known exact solution which is*

$$\mathbf{y}(t) = \begin{cases} y_1 = \sin(t) + 5\cos(t^2/2) \\ y_2 = \cos(t) + 5\sin(t^2/2) \\ y_3 = t \end{cases} .$$

If we compute the previous enclosures with the exact solution (by combining interval arithmetic with bisection method to obtain sharp results), we find:

- $\mathbf{y}(1.1) = ([5.003, 5.003]; [3.297, 3.297]; [1.1])$
- $\mathbf{y}([0.7, 0.8]) = ([[5.464, 5.494][1.977, 2.269]; [0.7, 0.8])$

The enclosure property of $R(t)$ and $\widetilde{R}(t)$ is then verified. ■

3 CSP with Differentials Constraints

Constraint Satisfaction Differential Problems (CSDP) extending the definition of NCSP have been defined in [7] by adding new variables to represent time derivative, *i.e.*, $\frac{dy}{dt}$ and a new kind of constraints of the form $\frac{dy}{dt} = f(y, t)$. The main drawback of CSDPs is that temporal properties cannot be encoded because the time variable t is not a variable of the problem. So, the property "for all t in $[0, 10]$, the system does not collide with an obstacle" cannot be represented as a CSDP in an obvious manner.

The work [8] considers differential equations as an implicit definition of temporal functions. In consequences, considering only solution function Φ of differential equations makes an easy embedding of differential equations into NCSP framework. Unfortunately, NCSPs naturally express existential problems, *i.e.*, problems of the form $\exists \mathbf{x} \in [\mathbf{x}], \bigwedge_{i=1}^{n} f_i(\mathbf{x}) = 0 \wedge \bigwedge_{i=1}^{m} g_i(\mathbf{x}) \leqslant 0$. While this approach is appealing to simply embed differential equations into constraint programming framework, it is also unable to express constraints where the time variable is a variable of the problem.

In both previous work [7] and [8], the main drawback is the lack of quantification on variables. A proposition of extension of these approaches to cope with this limitation is the main contribution of the paper.

In order to be able to define a CSDP which can answer a question of the type *"Will the system collide with an obstacle before ten seconds?"*, we propose a new definition of CSDP which mainly introduces quantification over variables.

Definition 2 (QCSDP) *Let S be a differential system as defined in* (2) *and $t_{end} \in \mathcal{R}_+$ the time limit of the problem handled. A QCSDP is a CSP defined by:*

- *a set of variables $\mathcal{V}$ including at least the time variable t, a vector $\mathbf{y}_0$ of dimension n representing the initial state-space of S, variables associated to the parameters $\mathbf{p}$, variables representing the control input $\mathbf{u}$ of S. We represent these variables by the vector $\mathbf{v}$;*
- *an initial domain $\mathcal{D}$ containing at least $[0, t_{end}]$ for time t, the initial domain $\mathcal{Y}_0$ of initial condition $\mathbf{y}_0$, $\mathcal{U}$ for control input $\mathbf{u}$, and $\mathcal{P}$ for parameters $\mathbf{p}$;*

- *a set of constraints* $\mathcal{C} = \{c_1, \ldots, c_e\}$ *composed of predicates over sets, that is, constraints of the form:*

$$c_i(\mathbf{v}) \equiv Q\mathbf{v} \in \mathcal{D}_i.f_i(\mathbf{v}) \diamond \mathcal{A}, \qquad \forall 1 \leqslant i \leqslant e,$$

with $Q \in \{\exists, \forall\}$, $f_i : \wp(\mathcal{R}^{|\mathcal{V}|}) \rightarrow \wp(\mathcal{R}^q)$ *stands for non-linear arithmetic expressions defined over variables* $\mathbf{v}$ *and solution of differential system* S, $\mathbf{y}(t; \mathbf{y}_0, \mathbf{p}, \mathbf{u}) \equiv \mathbf{y}(\mathbf{v})$. *The operator* $\diamond$ *denotes* $\{\subset, \supset, \cap_{=\emptyset}, \cap_{\neq\emptyset}\}$, *i.e., the set-based operations, and the set* $\mathcal{A} \subset \mathcal{R}^q$ *where* $q > 0$.

Note that $\cap_{=\emptyset}$ *stands for the predicate* $f(\mathbf{v}) \cap \mathcal{A} = \emptyset$ *and* $\cap_{\neq\emptyset}$ *stands for the predicate* $f(\mathbf{v}) \cap \mathcal{A} \neq \emptyset$. $\wp(\mathcal{E})$ *denotes the power-set of the set* $\mathcal{E}$, *while* $|\mathcal{E}|$ *denotes the cardinality of* $\mathcal{E}$.

Note that in Definition 2, constraints are expressed with set-based operator instead of equalities and inequalities to simplify the notations despite of the complete equivalence with equalities and inequalities. Moreover, set-based operations will emphasize, in Section 4, one important difficulty due to the representation of sets with respect to the correctness of the proposed approach.

As the QCSDP over sets is not computable in general, a set abstraction has to be used. Moreover, solving arbitrary quantified constraints over reals is a hard problem hence we focus on certain classes of problems related to the robotic domain. In particular, three main classes are considered. Problem 1 deals with the design of robot platform or control algorithms and for example is treated with a CSP approach in [19] or in [20]. Problem 2 deals with the design of robust control algorithms. It is the case study of Section 5 and it has been considered in the context of non-linear sampled switch systems in [21]. Problem 3 consists to find the subset of initial states such that there exists at least one evolution "viable", in the sense that at each time, the state of the evolution remains confined to the viability kernel [22]. The problem of viability kernel computation has been considered by the help of our method in [23].

Problem 1 (Appropriate design) *A parametric dynamics of a robot described by a differential system is considered, the parameters represent some possible choices in the characteristics of the robot such as its dimension, the power of the engine and so on. An* appropriate design problem *aims at finding a value of the parameters such as a set of functional requirements is fulfill. In other words, QCSDP problems are made of constraints over variables* $\mathbf{p}$, $\mathbf{u}$ *and* $\mathbf{y}_0$ *based on quantification* $\exists\mathbf{p}$, $\forall\mathbf{u}$, $\forall\mathbf{y}_0$, *and some temporal constraints (see Section 4).*

Problem 2 (Control synthesis) *A parametric dynamics of a robot described by a differential system is considered, the parameters represent perturbation. A* control synthesis problem *aims at finding value of the control* $\mathbf{u}$ *such that some requirements are fulfill taking into account all possible perturbations. In other words, QCSDP problems are made of constraints over variables* $\mathbf{p}$, $\mathbf{u}$ *and* $\mathbf{y}_0$ *based on quantification* $\exists\mathbf{u}$, $\forall\mathbf{p}$, $\forall\mathbf{y}_0$, *and some temporal constraints (see Section 4).*

Problem 3 (Viability kernel) *A parametric dynamics of a robot described by a differential system is considered, the parameters represent perturbation. A* viability kernel problem *aims at finding initial values of the system such that there exists at least one evolution remaining into the viability kernel taking into account all possible perturbations. In other words, QCSDP problems are made of constraints over variables* $\mathbf{p}$, $\mathbf{u}$ *and* $\mathbf{y}_0$ *based on quantification* $\exists\mathbf{y}_0$, $\forall\mathbf{p}$, $\exists\mathbf{u}$, *and some temporal constraints (see Section 4).*

Note that it is also possible to add a cost function in the Definition 2, and hence consider *constraint optimization problems.*

4 Reasoning Under Set Abstraction

In Definition 2, QCSDPs are defined over sets of values and hence they are not computable. Abstraction with boxes is considered in this section. By denoting $\alpha()$ the box abstraction, the constraints of QCSDP presented above becomes:

$$\begin{aligned} \alpha(c_i(\mathbf{v})) &\equiv \alpha(Q\mathbf{v} \in \mathcal{D}_i.f_i(\mathbf{v}) \diamond \mathcal{A}) && (6)\\ &\equiv Q\mathbf{v} \in [\mathbf{d}_i].\alpha(f_i(\mathbf{v}) \diamond \mathcal{A}) && (7)\\ &\equiv Q\mathbf{v} \in [\mathbf{d}_i].[f_i](\mathbf{v})\alpha(\diamond\mathcal{A}) \ . && (8) \end{aligned}$$

From Equation (6) to Equation (7), abstraction consists on an over-approximation of the domain of variables. Then, from Equation (7) to Equation (8), the interval enclosure is used as an abstraction of function f. It means that only enclosure of the solution of differential equations is considered, even if an

inner approximation could be available. Indeed, enclosure is mainly favored because the approach presented in this paper is oriented to safety problems and not to liveness. Abstraction $\alpha(\diamond\mathcal{A})$ is not distributive in a safe manner. Applied to the set $\mathcal{A}$, the function $\alpha \in \{\text{Hull}, \text{Int}\}$ stands for an interval abstraction function which maps a set $\mathcal{A}$ to a set of boxes. $\text{Hull}(\mathcal{A})$ stands for the interval over-approximation of the set $\mathcal{A}$ while $\text{Int}(\mathcal{A})$ stands for an interval representation (a single interval or a paving) of an under-approximation of the set $\mathcal{A}$. By an abuse of notation, $\diamond$ in Definition 3 will denote the abstraction of the set-theoretic operator $\diamond$ given in Definition 2 but it is important to keep in mind that the implementation is strongly dependent on the abstraction of the set $\mathcal{A}$.

The possible operations are gathered in the Table 1. Due to the abstraction, all these operators are weak. Nevertheless, some results after abstraction imply the original logic formula. In Table 1, question mark *"?"* means that neither true nor false implies a guaranteed result on the original logic formula. On the contrary, `true` means that if the abstract formula is valid, then the original one is valid, *i.e.*, $[f](\mathbf{v}) \subset \text{Int}(\mathcal{A}) \Rightarrow f(\mathbf{v}) \subset \mathcal{A}$. In the same way, `false` means that if the abstract formula is not valid, then the original one is not valid, *i.e.*, $\neg([f](\mathbf{v}) \cap_{\neq\emptyset} \text{Hull}(\mathcal{A})) \Rightarrow \neg(f(\mathbf{v}) \cap_{\neq\emptyset} \mathcal{A})$. Abstraction function $\alpha(\diamond\mathcal{A})$ can be distribute into $\alpha(\diamond)\alpha(\mathcal{A})$ by following carefully the results presented in Table 1.

Finally, we propose the Box-QCSDP formalism, which is a QCSP where sets are represented by boxes and is formally defined in Definition 3. It mainly introduces two abstractions: the abstraction of the solution of differential systems as defined in (2); and the abstraction of constraints c_i, in particular, the abstraction of the set $\mathcal{A}$.

The first and major abstraction allows us to manage with the differential system in a CSDP. It is performed by a validated simulation method as described in 2.3. This approach provides to us two abstraction functions guaranteed to enclose all the behaviors of the differential system.

The second abstraction is done to be able to handle the sets by an outer or an inner approximation, depending on the associated logic operation. In order to correctly translate a satisfaction problem, these capabilities offered by QCSDP have to be used in the right way.

Definition 3 (Box-QCSDP) *Let S be a differential system as defined in (2) and $t_{end} \in \mathcal{R}_+$ the time limit of the problem handled. A Box-QCSDP is defined by:*

- *a set of variables $\mathcal{V}$ including at least the time variable t, a vector $\mathbf{y}_0$ of dimension n representing the initial state-space of S, variables associated to the parameters $\mathbf{p}$, variables representing the control input $\mathbf{u}$ of S. We represent these variables by the vector $\mathbf{v}$;*
- *an initial box domain $[\mathbf{d}]$ containing at least $[0, t_{end}]$ for time t, the initial box $[\mathbf{y}_0]$ of initial condition $\mathbf{y}_0$, a box $[\mathbf{u}]$ for control input $\mathbf{u}$, and a box $[\mathbf{p}]$ for parameters $\mathbf{p}$;*
- *a set of constraints $\mathcal{C} = \{c_1, \ldots, c_e\}$ composed of predicates over sets, that is, constraints of the form*

$$c_i \equiv Q\mathbf{v} \in [\mathbf{d}_i].[f_i](\mathbf{v})\alpha\,(\diamond\mathcal{A})\,, \qquad \forall 1 \leqslant i \leqslant e,$$

with $Q \in \{\exists, \forall\}$, $[f_i] : \mathcal{IR}^{|\mathcal{V}|} \to \mathcal{IR}^q$ stands for an inclusion function of non-linear arithmetic expressions defined over variables $\mathbf{v}$ and an interval enclosure of the solution of differential system S, $[\mathbf{y}](t; [\mathbf{y}_0], [\mathbf{p}], [\mathbf{u}])$. The set theory based operations are expressed by $\diamond \in \{\subset, \supset, \cap_{=\emptyset}, \cap_{\neq\emptyset}\}$ and a set $\mathcal{A} \subset \mathcal{R}^q$ where $q > 0$. The function $\alpha(\diamond\mathcal{A})$ maps a part of logic operations into abstract ones, following Table 1.

Definition 3 is very dependent of the implementation. The following remarks give some insight to produce safe results.

Remark 1 *If a validated simulation cannot reach t_{end} and fails at $t = t_{fail}$ then $\widetilde{R}(t) = [-\infty, \infty]$, for all $t \geqslant t_{fail}$. Hence the enclosure of $[\mathbf{y}](t; [\mathbf{y}_0], [\mathbf{p}], [\mathbf{u}])$ is always correct and respects the contractor definition.*

Table 1: Available operations

			$\alpha(\mathcal{A})$	
			$\text{Int}(\mathcal{A})$	$\text{Hull}(\mathcal{A})$
$\alpha(f)$	$[f]$	$\subset$	`true`	?
		$\supset$	`false`	?
		$\cap_{=\emptyset}$	?	`true`
		$\cap_{\neq\emptyset}$	?	`false`

Table 2: Available operations

Verbal property	CSDP translation
Stay in $\mathcal{A}$	$\forall t \in [0, t_{\text{end}}]$, $R(t) \subseteq \text{Int}(\mathcal{A})$
In $\mathcal{A}$ at τ	$R(\tau) \subseteq \text{Int}(\mathcal{A})$
Has crossed $\mathcal{A}$	$\exists t \in [0, t_{\text{end}}]$, $R(t) \cap \text{Hull}(\mathcal{A}) \neq \emptyset$
Go out $\mathcal{A}$	$\exists t \in [0, t_{\text{end}}]$, $R(t) \cap \text{Hull}(\mathcal{A}) = \emptyset$
Has reached $\mathcal{A}$	$R(t_{\text{end}}) \cap \text{Hull}(\mathcal{A}) \neq \emptyset$
Finish in $\mathcal{A}$	$R(t_{\text{end}}) \subset \text{Int}(\mathcal{A})$

Remark 2 *Solutions of differential equations need to be pre-computed to obtain our abstraction functions $R(t)$ and $\widetilde{R}([t])$. If the parameters or control inputs involved in the definition of the differential equation is reduced by the CSDP solver, then the abstraction functions need to be computed again. It is a part of the propagation algorithm of a CSP.*

QCSDP, see Definition 2, allows one to describe a complex satisfaction problem in order to reason about a system defined by a differential equation. Nevertheless, a definition of the temporal constraints has not been done yet, it is the purpose of the end of this section. We consider the main interesting properties for robotic applications, by providing verbal operators in the sense of *"system has the property of"*, such as *"stay in till"*.

By convention, these Boolean operations provide a guaranteed true answer. It means that *"true"* is proven while *"false"* means *"uncertain"* due to the abstraction of boxes. In table 2, the different available verbal operators and their Box-QCSDP translation are gathered. Despite this list of specifications is limited it emphasizes the main difficulty to reason on abstraction of the behaviors of robots.

From table 2 and due to the abstraction, *"Go out"* is not equivalent to $\neg$(*"Stay in"*). Operators *"Has reached"* and *"Has crossed"* are poor tests (true also means uncertain). They are equivalent but the validity of *"Has reached"* at the end of a simulation implies a notion of stability. On the other hand, the negation of these latter operators are strong and are useful to speed up a Box-QCSDP solver.

Following the same generalization than for Constraint Optimization Problems based on CSP, a cost function can be added to Box-QCSDP to perform optimization.

Definition 4 (Optimization on Box-QCSDP) *A Box-Quantified Constraint Optimization Differential Problem involves in addition of a Box-QCSDP an objective function $Q(v_1, \ldots, v_n)$ that has to be maximized or minimized over the set of all feasible solutions.*

The details on algorithms to solve optimization problems is out of the scope of this paper, but there are the same than the ones used for CSP [24].

Box-QCSDP and this list of specifications have been implemented in a library named DynIBEX[1] [25].

5 Case Study: Robot Path Planning

We present an application of Box-QCSDP based on the algorithm Box-RRT defined in [26] which aims at producing a robust path for a car-like vehicle while avoiding obstacles.

The dynamic model of the car is defined by

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \dfrac{v}{L}\tan(\delta + \varepsilon) \end{cases}. \tag{9}$$

The state of the car is defined by the car position (x, y) and its heading θ. L is the distance between the front and rear wheels. There are two control inputs v for the longitudinal speed and $\delta \in [\delta_{\min}, \delta_{\max}]$ for the steering angle. The noise element $\varepsilon = [-0.001, 0.001]$ stands for the steering angle imprecision. A second source of uncertainties is the initial position of the vehicle. We denote by $\text{car}(t; \mathbf{s}_0)$ the solution of Equation (9) and by $[\text{car}](t; \mathbf{s}_0)$ its interval enclosure produce by guaranteed integration methods. We denote by $s_0 = (x_0, y_0, \theta_0)$ and $[s_0] = [x_0] \times [y_0] \times [\theta_0]$ the state variables and the initial conditions. We denote by $[\mathbf{s}_f]$ the targeted state-space.

Box-RRT algorithm extends Rapid Random Tree (RRT) algorithm with interval analysis tools. A brief recall of the Box-RRT algorithm is given before introducing our formalization with Box-QCSDP. From a

[1] `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

map, denoted by Map, represented by a paving, *i.e.*, a set of non-overlapping boxes and from a list of obstacles, denoted by $\mathcal{L}_a$, Box-RRT looks for a safe path between $[\mathbf{s}_0]$ and $[\mathbf{s}_f]$, avoiding all obstacles in $\mathcal{L}_a$ and staying in Map. To do this, a tree T of possible paths is generated from a random walk inside the state-space represented by a paving. Box-RRT is an iterative algorithm which mainly follows three steps until $[\mathbf{s}_f]$ is reached

i. pick a random state $[\mathbf{s}_{\text{random}}]$,

ii. find the closest state of $[\mathbf{s}_{\text{random}}]$ in the tree T named $[\mathbf{s}_{\text{nearest}}]$,

iii. pick a control $\mathbf{u}$ and compute $[\mathbf{s}_{\text{new}}]$ which is the final state of the trajectory starting from $[\mathbf{s}_{\text{nearest}}]$ with the control $\mathbf{u}$. If this trajectory reaches an obstacle, an other state is chosen. Otherwise if $[\mathbf{s}_{\text{new}}] \subset [\mathbf{s}_f]$ then a path is found and the algorithm terminates. Otherwise, the target $[\mathbf{s}_f]$ has not been reached and $[\mathbf{s}_{\text{new}}]$ is a possible safe state being on the path to reach the target $[\mathbf{s}_f]$. Hence, $[\mathbf{s}_{\text{new}}]$ is added as a new node in T and the exploration of the state-space continues.

In Box-RRT algorithm, Step 3 is a Box-QSCDP. In particular, there are several constraints that must be taken into account while the path is generated. First, the path generation works on a finite dimension map and so trajectories must stay inside this map. Obviously, a constraint expressing the avoidance of obstacles shall be considered. An other constraint has to be defined to detect when a trajectory reaches the targeted region of the state-space. In consequence, the path generation is translated into Box-QCSDP which is defined in Equation (10). Note that this Box-QCSDP has to be solved at each iteration of the Box-RRT algorithm until a path has been found.

$$
\begin{aligned}
&\mathcal{V} = \{t, x_0, y_0, \theta_0, v, \delta\} \ , \\
&\mathcal{D} = \{[t] = [0,1], [x_0] = [0.0, 0.3], [y_0] = [0.0, 0.3], \\
&\qquad\quad [\theta_0] = [0.0]\} \ , \\
&\mathcal{C} = \{ \\
&\quad c_1 = \forall \mathbf{s}_0 \in [\mathbf{s}_0], \forall t \in [t] \cdot [\text{car}](t; \mathbf{s}_0) \subset \text{Map}, \\
&\quad c_2 = \forall \mathbf{s}_0 \in [\mathbf{s}_0], \forall t \in [t], \forall \mathbf{s}_a \in [\mathcal{L}_a] \cdot [\text{car}](t; \mathbf{s}_0) \cap_{\neq \emptyset} \mathbf{s}_a, \\
&\quad c_3 = \forall \mathbf{s}_0 \in [\mathbf{s}_0], \exists t \in [t] \cdot [\text{car}](t; \mathbf{s}_0) \subset [\mathbf{s}_f] \\
&\} \ .
\end{aligned}
\tag{10}
$$

In Equation (10), the constraint c_1 states that the trajectory $[\text{car}](t)$ from $[\mathbf{s}_{\text{nearest}}]$ to $[\mathbf{s}_{\text{new}}]$ has to stay inside the map Map. The constraint c_2 states that the trajectory $[\text{car}](t)$ shall not intersect an obstacle and finally the constraint c_3 states that a path is found if one state of trajectory is included into $[\mathbf{s}_f]$.

An example of the exploration of the state-space produced by the Box-RRT algorithm is given in Figure 3. The red squares stand for obstacles while the blue square represents $[\mathbf{s}_f]$. The small black square in the center of the figure is $[\mathbf{s}_0]$ while the black square inside $[\mathbf{s}_f]$ is the final state of the trajectory reaching the target.

6 Conclusion

In this paper, we presented a framework for the formal verification of robotic behaviors in presence of uncertainties. Based on the constraint satisfaction problem formalism and by the help of interval tools such as interval arithmetic and the validated integration of differential equations, a method to verify the safety, in a guaranteed manner, has been developed. Our approach mainly exploits box abstraction of uncertainties, sets, functions and differential equations. Box abstraction is a very powerful approach to manage with uncertainties and, more particularly, with differential equations. Nevertheless, this abstraction introduces another difficulty, which is the handling of logic formula, because some logic operators become weak. We carefully analyzed available operators and a new formalism called Box-QCSDP has been finally presented. A well known problem, the robot path planning, has been studied under the formalism of Box-QCSDP. An algorithm based on Rapid-exploring Random Tree has been defined including constraints (obstacles and objectives) and solved with our method. The resulting path is guaranteed by construction, considering the robot dynamics, which is an interesting improvement of classical path planners. To conclude, we presented in this paper a new formalism that allows one to solve complex problems linking dynamical systems and temporal logic, in a validated way.

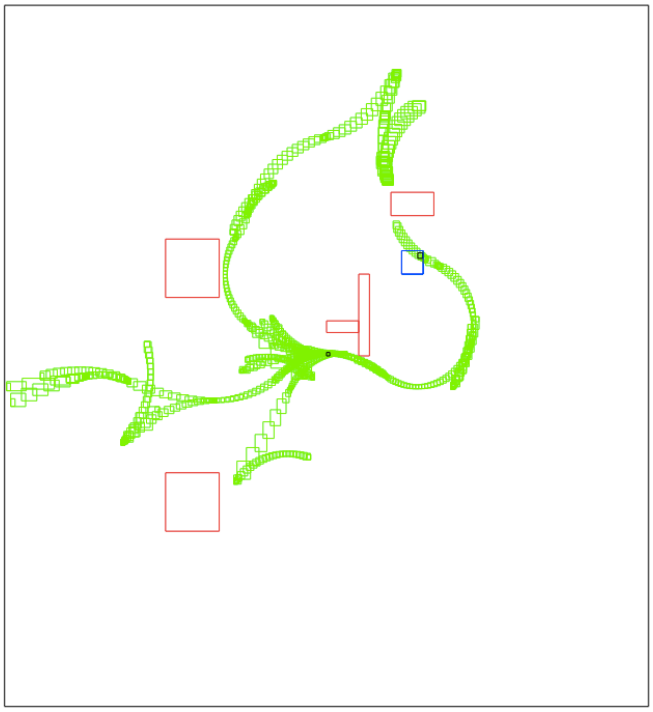

Figure 3: Results of Box-RRT

Acknowledgment

The authors are grateful to François Pessaux for his pertinent comments on the early versions of this work.

This research benefited from the support of the "Chair Complex Systems Engineering - Ecole Polytechnique, THALES, DGA, FX, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech and FDO ENSTA". This research is also partially funded by DGA MRIS.

References

[1] Michel Rueher. Solving continuous constraint systems. In *International Conference on Computer Graphics and Artificial Intelligence*, 2005.

[2] Nedialko S. Nedialkov, K. Jackson, and Georges Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.

[3] Andreas Rauh, Michael Brill, and Clemens GüNther. A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP. *International Journal of Applied Mathematics and Computer Science*, 19(3):381–397, 2009.

[4] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing electronic edition*, 22, 2016.

[5] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Simulation of Differential Algebraic Equations with Runge-Kutta Methods. *Reliable Computing electronic edition*, 22, 2016.

[6] Micha Janssen, Yves Deville, and Pascal Van Hentenryck. Multistep filtering operators for ordinary differential equations. In *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 1999.

[7] Jorge Cruz and Pedro Barahona. Constraint reasoning over differential equations. *Applied Numerical Analysis and Computational Mathematics*, 1(1):140–154, 2004.

[8] Alexandre Goldsztejn, Olivier Mullier, Damien Eveillard, and Hiroshi Hosobe. Including ordinary differential equations based constraints in the standard CP framework. In *Principles and Practice of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 2010.

[9] Bardh Hoxha and Georgios E. Fainekos. Planning in dynamic environments through temporal logic monitoring. In *AAAI Workshop: Planning for Hybrid Systems*, volume WS-16-12 of *AAAI Workshops*. AAAI Press, 2016.

[10] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M Murray. Synthesis of reactive switching protocols from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 58(7):1771–1785, 2013.

[11] Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *Automated Technology for Verification and Analysis*, volume 5311 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2008.

[12] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer, 2013.

[13] Ramon E. Moore. *Interval Analysis*. Series in Automatic Computation. Prentice Hall, 1966.

[14] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.

[15] Yahia Lebbah and Olivier Lhomme. Accelerating filtering techniques for numeric CSPs. *Journal on Artificial Intelligence*, 139(1):109 – 132, 2002.

[16] F. Benhamou, D. McAllester, and P. Van Hentenryck. Clp (intervals) revisited. Technical report, Providence, RI, USA, 1994.

[17] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[18] Olivier Lhomme. Consistency techniques for numeric CSPs. In *International joint conference on Artifical intelligence*, pages 232–238, 1993.

[19] Julien Alexandre dit Sandretto, Douglas Piccani de Souza, and Alexandre Chapoutot. Appropriate design guided by simulation: An hovercraft application. In *Model-Driven Robot Software Engineering*. ACM, 2016.

[20] Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier. Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis. In *Synthesis of Complex Parameters*, volume 44 of *OpenAccess Series in Informatics (OASIcs)*, pages 91–102. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[21] Adrien Le Coent, Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Laurent Fribourg. Control of Nonlinear Switched Systems Based on Validated Simulation. In *Symbolic and Numerical Methods for Reachability Analysis*, Vienna, Austria, 2016.

[22] Jean-Pierre Aubin. A survey of viability theory. *SIAM Journal on Control and Optimization*, 28(4):749–788, 1990.

[23] Dominique Monnet, Luc Jaulin, Jordan Ninin, Julien Alexandre dit Sandretto, and Alexandre Chapoutot. Viability kernel computation based on interval methods. In *Small Workshop on Interval Methods*, Prague, Czech Republic, 2015.

[24] Eldon R. Hansen. *Global Optimization Using Interval Analysis*. CRC Press, 2 edition, 2003.

[25] Julien Alexandre dit Sandretto and Alexandre Chapoutot. DynBEX: a Differential Constraint Library for Studying Dynamical Systems (Poster). In *Conference on Hybrid Systems: Computation and Control*, 2016.

[26] Romain Pepy, Michel Kieffer, and Eric Walter. Reliable robust path planning with application to mobile robots. *International Journal of Applied Mathematics and Computer Science*, 19(3):413–424, 2009.