Improving validated computation of Viability Kernels

Benjamin Martin
LIX, École Polytechnique, Université Paris-Saclay
Palaiseau, France
bmartin@lix.polytechnique.fr

Olivier Mullier
U2IS, ENSTA ParisTech, Université Paris-Saclay
Palaiseau, France
olivier.mullier@ensta-paristech.fr

ABSTRACT

The study of viability kernels can be of critical importance for the verification of control systems. A viability kernel over a set of safe states is the set of initial states for which the trajectory can be controlled so as to stay within the safe set for an indefinite amount of time. This paper investigates improvements of the rigorous method from Monnet *et al.* [19, 20]. This method computes an inner-approximation of the viability kernel of a continuous time control system using methods based on interval analysis. It consists of two phases: first an initial inner-approximation of the viability kernel is computed via Lyapunov-like functions; second the initial inner-approximation is improved by finding other states that can reach the inner-approximation, without exiting the safe set, using validated numerical integration. Among the improvements, we discuss an approach inspired by an interval method using barrier functions for computing a good initial inner-approximation of the viability kernel, easing the improvement phase.

KEYWORDS

Viability Kernel, Interval Analysis, Validated Numerical Integration, Lyapunov-like Functions

ACM Reference Format:
Benjamin Martin and Olivier Mullier. 2018. Improving validated computation of Viability Kernels. In *HSCC '18: 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), April 11–13, 2018, Porto, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178126.3178141

1 INTRODUCTION

The viability theory aims at studying the evolution of controlled dynamics occurring in many domains (*e.g.*, in chemistry, biology or physics). The main focus on viability theory [3] is the computation of the viability kernel of controlled dynamics that is the subset of initial states such that there exists at least one evolution of the system viable in this subset.

The viability kernel helps to identify critical states of the system for which there are no known control for avoiding unsafe states, or states for which it is possible to evolve inside a safe set for indefinite amount of time. We can identify different approaches from the literature that either approximate the viability kernel [7, 22, 27], or propose a validated outer and/or inner-approximation of it [14, 16, 28]. In this paper, we focus on a recently proposed approach by Monnet *et al.* [19, 20] for computing validated inner-approximation[1] of the viability kernel by means of a two phase interval based approach. We study in particular improvements of the first phase which aims at computing an initial inner-approximation of the viability kernel via Lyapunov-like functions. The second phase improves the initial inner-approximation through the computation of a capture basin of the sets obtained in the first phase. Our goal is then to develop a new technique that improves the result of the first phase to ease the computational burden of the second phase.

Related work. When considering low dimensional systems, many methods from the literature embrace a discretization of the state space by a grid, see e.g. [7, 22, 27]. Reasoning on such grid makes the problem of computing a viability kernel more tractable while remaining accurate. They usually fail to provide strict guarantees on the numerical computations. However, some approaches have proposed validated computations of inner-approximation of viability kernel via discretization/abstraction. In [10], approximate bisimilar abstractions of systems is used to derive safe control laws that constrain every trajectories inside a domain. Building such abstractions require additional stability assumptions on the system, and continuous time systems require time discretization. Validations can then only occur on the initial and final

[1]Outer-approximations can also be obtained using [19, 20] but this is not covered in this paper.

states of the concrete abstracted system at each time step, and not on the trajectory itself. In [26], the notion of feedback refinement controller is used for building a controller on abstracted systems, via discretization by means of hyper-intervals and outer-approximations, that meets the specification of the concrete system. Again, the time is discretized and viability conditions are only checked at the end of each steps.

The approach from [20] which we follow can be viewed as a rigorous version of griding approaches. The state space is decomposed into a paving with boxes (cartesian product of intervals), for which validated numerical techniques are further applied to decide whether a box belongs to the viability kernel of the system. Validated numerical integration is used to build tubes enclosing all trajectories starting from a box, which in turn can be used to detect collision with the boundary of the set of safe states.

For higher-dimensional problems, several alternatives have been proposed. We can note Lagrangian methods like [14] for linear systems, in which a piece-wise ellipsoidal rigorous approximation of the viability kernel is computed. The method is however limited to linear systems and only a finite time horizon is considered for the computed viability kernel. For polynomial systems, the approach proposed in [28] consists in building sequence of Lyapunov-like polynomial functions. They are based on relaxations of the problem of satisfying Lyapunov-like conditions by means of sum of square polynomials, written as Bilinear Matrix Inequality problems. Contrary to our present work, the authors did not consider any input for the system.

In [16] the authors proposed an approach for computing maximal controlled invariant sets (equivalent to the notion of viability kernel) within a semi-algebraic set for controlled polynomial systems. The problem is written as an infinite dimensional linear program solved by a hierarchy of linear matrix inequality problems. Although presented for discrete-time system, the method can be extended to continuous systems. Apart from the limitations to polynomial systems, both state and input space must be in the form of a conjunction of algebraic inequalities. The method we follow [20] and improve in this paper can consider a broader type of constraints for the state space. The method from [16] is nevertheless a possibly complementary approach to ours, in particular for the first phase of the algorithm we are improving in this paper.

Eventually, on interval-based approaches, the work from [11] proposes to tackle a specific class of Exist-ForAll quantified constraint systems. The problem we need to solve for the first phase of the method described in this paper differs but almost falls into this class of quantified constraint systems.

2 BACKGROUND

2.1 Problem formulation

In this paper, we consider the computation of the viability kernel [3] of controllable dynamical systems of the form

$$(S)\begin{cases}\dot{\mathbf{x}} = f(\mathbf{x},\mathbf{u})\\ \mathbf{x}(0) = \mathbf{x}_0\end{cases} \tag{1}$$

with $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ a non-linear globally Lipschitz continuous function, for all $t > 0$, $\mathbf{u}(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $\mathbf{x}(t) \in \mathbb{R}^n$ the state. For simplification, we will denote $\mathbf{u} \in \mathcal{U}$ instead of $\mathbf{u}(t) \in \mathcal{U}, \forall t$.

For a given function $h(.) : \mathbb{R}^n \to \mathbb{R}$, its time derivative with respect to the system (1) is denoted by:

$$\dot{h}(\mathbf{x},\mathbf{u}) := \langle \nabla h(\mathbf{x}), f(\mathbf{x},\mathbf{u})\rangle, \tag{2}$$

where $\nabla h(\mathbf{x})$ denotes the gradient of h and $\langle .,. \rangle$ the standard inner product of two vectors. The time derivative is also often called *Lie derivative*.

For a system such as (1), the viability kernel of a set $K \in \mathbb{R}^n$ corresponds to the set $\text{Viab}_S(K)$ such that

$$\text{Viab}_S(K) = \{\mathbf{x}_0 \in K | \exists \mathbf{u} \in \mathcal{U}, \forall t \geqslant 0, \varphi(\mathbf{x}_0,\mathbf{u},t) \in K\}, \tag{3}$$

with $\varphi : \mathbb{R}^n \times \mathcal{U} \times \mathbb{R}^+ \to \mathbb{R}^n$, the solution operator of (1) *i.e.* the function such that $\varphi(\mathbf{x}_0,\mathbf{u},t)$ is the value of $\mathbf{x}(t)$ when $\mathbf{x}(0) = \mathbf{x}_0$ and the control $\mathbf{u}$ is applied. The viability kernel is here considered for an unbounded time ($t \in [0, +\infty[$).We will assume that K is bounded, closed, and defined by a boolean combination (conjunction and disjunction) of inequalities.

The basis of the method from Monnet *et al.* [20] uses another notion from viability theory that is the capture basin. The capture basin of a set T, viable in K in a bounded time horizon t_{end}, denoted $\text{Capt}_S^{t_{\text{end}}}(K,T)$ is given by the following:

$$\text{Capt}_S^{t_{\text{end}}}(K,T) = \left\{ \mathbf{x}_0 \in K \;\middle|\; \begin{array}{c} \exists \tilde{t} \in [0, t_{\text{end}}], \exists \tilde{\mathbf{u}} \in \mathcal{U}, \\ \varphi(\mathbf{x}_0, \tilde{\mathbf{u}}, \tilde{t}) \in T \\ \wedge \forall t \in \left[0, \tilde{t}\right], \varphi(\mathbf{x}_0, \tilde{\mathbf{u}}, t) \in K \end{array} \right\}. \tag{4}$$

This capture basin is the set of initial conditions in K for which there exist an evolution that allow in the bounded time t_{end} to reach a given target T while remaining in K.

The following proposition links capture basin to viability kernels.

Proposition 2.1. *Given a set $T \subseteq Viab_S(K)$, then*

$$\text{Capt}_S^{t_{end}}(K,T) \subseteq Viab_S(K), \ \forall t_{end} \geq 0. \tag{5}$$

In other words, the (finite-time horizon) capture basin of any subset of the viability kernel is also part of the viability kernel. Proposition 2.1 is used in [20] to produce iteratively an inner-approximation of the viability kernel $\text{Viab}_S(K)$.

2.2 Interval analysis

Interval analysis (IA) is a branch of numerical analysis born in the 60s [21]. The key idea of interval analysis is to replace real (in practice floating point number) computations by interval ones that always contain the correct result. It can be used for both monitoring numerical rounding errors and set computations with applications in Global optimization and Constraint Programming. We refer to [13, 15, 24] for a broad overview of IA.

An interval $[x] = [\underline{x}, \overline{x}] \subset \mathbb{R}$ is a closed subset of reals defined by a lower bound $\underline{x} \in \mathbb{R}$ and upper bound $\overline{x} \in \mathbb{R}$ such that $\underline{x} \leqslant \overline{x}$. We denote by $\mathbb{IR}$ the set of all intervals of reals. A n-dimensional box $[\mathbf{x}]$ is a cartesian product of n intervals $([\mathbf{x}])_{1\leq i\leq n}$, defined by a lower and an upper bound vector $\underline{\mathbf{x}}$ and $\overline{\mathbf{x}}$. Given an interval $[x]$, $\text{mid}[x] := 0.5(\underline{x} + \overline{x})$ is its *center*, $\text{wid}[x] := \overline{x} - \underline{x}$ is its *width*. The width of a box is the maximum of its component-wise widths.

An *interval extension* $[f] : \mathbb{IR}^n \to \mathbb{IR}$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ is an interval function satisfying the containment principle, i.e. $f([\mathbf{x}]) := \{f(\mathbf{x}) : \mathbf{x} \in [\mathbf{x}]\} \subseteq [f]([\mathbf{x}])$ for any box $[\mathbf{x}] \in \mathbb{IR}^n$. The *natural extension* $[f]$ of f, which will be considered as the default interval extension in the paper, consists in replacing all arithmetic and unary operations in the function f by their interval arithmetic counterparts (each satisfying the containment principle).

A *contractor* [6] $C_c : \mathbb{IR}^n \to \mathbb{IR}^n$, where c is a real arithmetic property $\mathbf{g}(\mathbf{x}) \subseteq \mathcal{A}$ with $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^k$ and $\mathcal{A} \subset \mathbb{R}^k$, is an interval operation such that $\forall [\mathbf{x}] \in \mathbb{IR}^n$, $C_c([\mathbf{x}]) \subseteq [\mathbf{x}]$ and $\mathbf{g}([\mathbf{x}]) \cap \mathcal{A} = \mathbf{g}(C_c([\mathbf{x}])) \cap \mathcal{A}$. As a consequence, for any $\mathbf{x} \in [\mathbf{x}] \setminus C_c([\mathbf{x}])$, then $\mathbf{g}(\mathbf{x}) \notin \mathcal{A}$. Contractors are useful for reducing boxes toward the solutions to constraint systems.

Eventually, consider a system of equation $\mathbf{h}(\mathbf{x}) = \mathbf{0}$, with $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^k$ and $k \leq n$. Consider also a partition of the vector $\mathbf{x} = (\mathbf{y}, \mathbf{z})$ with $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{z} \in \mathbb{R}^k$ $(m = n - k)$, and the analogous partition for boxes in $\mathbb{IR}^n$. A (parametric) interval Newton operator $\mathcal{N} : \mathbb{IR}^m \times \mathbb{IR}^k \to \mathbb{IR}^k$ is an operator satisfying: $\forall \mathbf{z} \in [\mathbf{z}]$, $(\exists \mathbf{y} \in [\mathbf{y}],\ \mathbf{h}(\mathbf{y}, \mathbf{z}) = \mathbf{0}) \implies \mathbf{z} \in \mathcal{N}([\mathbf{y}], [\mathbf{z}])$, and $\mathcal{N}([\mathbf{y}], [\mathbf{z}]) \subset [\mathbf{z}] \implies \forall \mathbf{y} \in [\mathbf{y}],\ \exists \mathbf{z} \in [\mathbf{z}],\ \mathbf{h}(\mathbf{y}, \mathbf{z}) = \mathbf{0}$. Well known interval operator are the Hansen-Sengupta and the Krawczyk operator [24]. Generally used as a contractor, an interval Newton operator can also be used for the search of a certified enclosure of solutions to a system of equations when coupled with inflation techniques, see e.g. [12].

2.3 Validated numerical integration

Validated numerical integration is designed to produce rigorous results on the solution of Initial Value Problem with Ordinary Differential Equations (IVP-ODES). Most techniques, since Ramon Moore's seminal work [21], are based on Taylor series [23]. More recent work [4, 5, 9, 17, 18] deals with the adaptation of Runge Kutta methods for ODES and Differential Algebraic Equations, see [2].

Validated numerical integration is a two step procedure: *a priori* enclosure on a time interval $[0, t_{\text{end}}]$ and a tighter approximation at a given time t_{end} (see Figure 1). Considering the system (S) from (1), validated numerical integration methods produce an outer-approximation of the set $\{\varphi(\mathbf{x}_0, \mathbf{u}, t_{\text{end}}) \mid \mathbf{x}_0 \in [\mathbf{x}_0],\ \mathbf{u} \in \mathcal{U}\}$, the set of values $\mathbf{x}(t_{\text{end}})$ can take when considering $\mathbf{x}(0) \in [\mathbf{x}_0]$ and $\mathbf{u} \in \mathcal{U}$. It allows also to produce an outer-approximation of $\{\varphi(\mathbf{x}_0, \mathbf{u}, t) \mid \mathbf{x}_0 \in [\mathbf{x}_0],\ \mathbf{u} \in \mathcal{U}, t \in [0, t_{\text{end}}]\}$ representing the set of all trajectories starting in $[\mathbf{x}_0]$, $\mathbf{u} \in \mathcal{U}$ and finishing at time t_{end}.

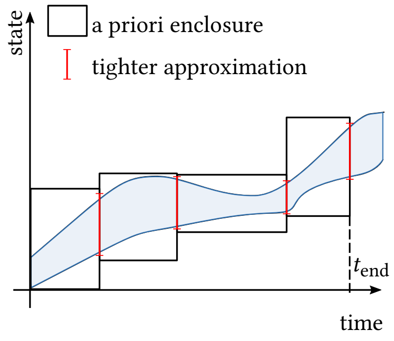

Figure 1: The two steps of a validated numerical integration.

3 MAIN RESULTS

In this paper, we consider the general framework from Monnet *et al.* [20] and investigate improvements of this interval based approach. First, we use interval methods to compute barrier-like functions as in [8], which generalize the first stage of the algorithm. The aim is to provide an initial inner-approximation of $\text{Viab}_S(K)$ that is as large as possible in order to ease the improvement phase.

3.1 First inner approximation

The original approach based on Lyapunov-like functions for computing the initial inner-approximation of the viability kernel in [20] has the advantage of being simple and fast. However, it does not focus on finding an initial inner-approximation that is as large as possible. Taking more time computing a larger inner-approximation is relevant for improving the second phase of the method which can be very time consuming. A better trade-off can hence be gained by taking more time computing a better inner-approximation during the first-phase.

Controlled barrier functions. As in [20], we want to compute a continuously differentiable function satisfying the following properties:

$$\forall \mathbf{x} \in K \; (\exists \mathbf{u} \in \mathcal{U},\; h(\mathbf{x}) = 0 \implies \dot{h}(\mathbf{x}, \mathbf{u}) < 0) \quad (6)$$
$$\wedge \; (\mathbf{x} \in \partial K \implies h(\mathbf{x}) > 0) \quad (7)$$

where ∂K denotes the boundary of K. The condition (7), not present in [20], implies that the set $\mathcal{H} := \{\mathbf{x} | h(\mathbf{x}) \leq 0\} \cap K \subset \text{int } K$ - where int K denotes the interior of K - and hence that the condition (6) is actually a valid (controlled) barrier certificate of $\mathcal{H}$ provided it is not empty. In other words, for any initial condition $\mathbf{x}_0 \in \mathcal{H}$, there is always a control leading the evolution to remain in $\mathcal{H}$, which further implies that $\mathcal{H} \subseteq \text{Viab}_S(K)$. We can also see the function h as a Lyapunov-like function for which we have leveraging positiveness and restricted the decrease of h on the boundary of $\mathcal{H}$.

Proposition 3.1. *Given a function h that satisfies* (6) *and* (7)*, then the set* $\mathcal{H} := \{\mathbf{x} | h(\mathbf{x}) \leq 0\} \cap K$ *is included in* $Viab_S(K)$.

Proof. Suppose the non-trivial case $\mathcal{H}$. From (7), we have that $\mathcal{H}$ is included in int K. Since K is bounded, so is $\mathcal{H}$. Finally, since h is continuous then $\mathcal{H}$ is closed. Theorem 3.1 from [20] can then be applied as (6) is equivalent to the notion of V-viability defined in [20]. This theorem entails that $\mathcal{H} \subseteq \text{Viab}_S(K)$. □

In [20], convex quadratic functions satisfying (6) are obtained by computing Lyapunov function of the linearized system around equilibrium points of (S) in (1). It is a fast method that do not intend to build a wide inner-approximation of the viability kernel.

We instead consider parametric template functions, similar to what has been investigated in [8], in order to find such functions h. Given a parametric template function $h_{\mathbf{p}}$, the problem is then to find parameters $\mathbf{p} \in \mathcal{P} \subset \mathbb{R}^k$ satisfying the following quantified constraints (equivalent to (6) and (7)):

$$\forall \mathbf{x} \in K \; (\exists \mathbf{u} \in \mathcal{U},\; h_{\mathbf{p}}(\mathbf{x}) \neq 0 \quad \vee \quad \dot{h}_{\mathbf{p}}(\mathbf{x}, \mathbf{u}) < 0) \quad (8)$$
$$\wedge \; (\mathbf{x} \notin \partial K \quad \vee \quad h_{\mathbf{p}}(\mathbf{x}) > 0) \quad (9)$$

We will assume next that $\mathcal{P}$ and $\mathcal{U}$ are boxes. The approach presented in [8] is referred as *Computable Sufficient Condition - Feasible Point Searcher* (CSC-FPS) method. The FPS loop organizes the search for valid parameter while the CSC procedure effectively attempt to validate, or reject, parameters. It is used to find a valid barrier functions for dynamical systems separating trajectories starting from a set of initial conditions from an unsafe set of states. The templates in [8] require then to satisfy some additional constraints ensuring this separation. Note also that no control is considered in [8]. In order to deal with the control functions, we suggest to relax the constraint (8) as follows. Given a finite set of control functions $U := \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_q\}$, where for each $i = 1, \ldots, q,\; \mathbf{u}_i \in \mathcal{U}$, the constraint

$$\forall \mathbf{x} \in K,\; h_{\mathbf{p}}(\mathbf{x}) \neq 0 \bigvee_{i=1}^{q} \dot{h}_{\mathbf{p}}(\mathbf{x}, \mathbf{u}_i) < 0, \quad (10)$$

is a stricter, but easier to check, constraint than (8).

The Best-FPS-CSC algorithm. We suggest to adapt the FPS-CSC algorithm from [8] for computing such barrier functions $h_{\mathbf{p}}$. Similar to [8] we denote by ξ the constraint system corresponding to (10) and (9), i.e.:

$$\xi(\mathbf{p}, \mathbf{x}) := \left(h_{\mathbf{p}}(\mathbf{x}) \neq 0 \bigvee_{i=1}^{q} \dot{h}_{\mathbf{p}}(\mathbf{x}, \mathbf{u}_i) < 0 \right) \wedge \left(\mathbf{x} \notin \partial K \vee h_{\mathbf{p}}(\mathbf{x}) > 0 \right). \quad (11)$$

The CSC-FPS approach aims at finding a parameter vector $\mathbf{p} \in \mathcal{P}$ such that $\forall \mathbf{x} \in K,\; \xi(\mathbf{p}, \mathbf{x})$. It also requires tests to prove that a given parameter box $[\mathbf{p}]$ does not contain any valid parameter for ξ. For clarity, we also give the expression of the negation of ξ:

$$\neg\xi(\mathbf{p}, \mathbf{x}) := \left(h_{\mathbf{p}}(\mathbf{x}) = 0 \bigwedge_{i=1}^{q} \dot{h}_{\mathbf{p}}(\mathbf{x}, \mathbf{u}_i) \geq 0 \right) \vee \left(\mathbf{x} \in \partial K \wedge h_{\mathbf{p}}(\mathbf{x}) \leq 0 \right). \quad (12)$$

Proving the non-validity of a box $[\mathbf{p}]$ requires to show that $\forall \mathbf{p} \in [\mathbf{p}],\; \exists \mathbf{x} \in K,\; \neg\xi(\mathbf{p}, \mathbf{x})$. It is then necessary to handle proof of solution to system of equations like $h_{\mathbf{p}}(\mathbf{x}) = 0$, which is not considered in [8]. However, constraint systems similar to (12) are rigorously treated in [12]. To this end, they have proposed to use parametric interval Newton tests to search for the existence of such solutions $\mathbf{x}$. We thus integrate such tests to the CSC-FPS algorithm.

Eventually, the original CSC-FPS algorithm [8] stops once a valid parameter $\mathbf{p} \in \mathcal{P}$ is found. Therefore, we additionally embed the FPS algorithm into an optimization loop, leading to the Best-FPS algorithm.This algorithm corresponds to a particular Branch & Bound algorithm which minimizes an objective function over the parameters subject to the universally quantified constraints defined by (8)-(9). The objective function $v : \mathbb{R}^k \to \mathbb{R}$ to optimize must correspond to the maximization of the volume of $\mathcal{H}$ - or as we will consider afterwards, the minimization of the negation of this volume. The expression of such an objective is complex if we consider a general template function, and may even not be available as such. For specific templates however, some simple objective function can be used (see Section 4). Additional constraints on the parameters $\mathbf{p}$ can also be considered, for e.g. constraining $h_{\mathbf{p}}$ to be a convex function. We denote this constraint system $\psi : g(\mathbf{p}) \in \mathcal{A}$, with $g : \mathbb{R}^k \to \mathbb{R}^l$ and $\mathcal{A} \subset \mathbb{R}^l$. Contractors on such a constraint system can be provided.

Algorithm 1 describes the Best-FPS algorithm. It takes as input the parameter box domain $\mathcal{P}$, a box $[\mathbf{x}]_{\text{init}}$ enclosing K, the constraint systems ψ (if any) and ξ, the objective function v and a precision $\epsilon_{\mathbf{p}} > 0$. The main loop iterates on parameter boxes that are stored in a heap structure $\mathcal{L}$, initially containing the whole box $\mathcal{P}$ at line 1. The best valid parameter $\hat{\mathbf{p}}$ and its objective valuation are initialized to dummy values, meaning no viable parameter has yet been found. The heap we suggest to use is a double heap as in [25], i.e. two heaps whose top element are lowest with respect to two different criterion. The top of one heap is the box parameter $[\mathbf{p}]$ whose lower bound on the objective function - simply obtained as $\min{[v]}([\mathbf{p}])$ - is the lowest, meaning that it prioritize best parameters $\mathbf{p}$. The other is the box $[\mathbf{p}]'$ with the highest upper bound on the objective function - obtained analogously- which focuses on parameters that are likely to satisfy the quantified constraint system. The choice of the top of which heap to select the next box to iterate at line 4 is random. Although the first sorting criterion is standard for its purpose, the choice of the second one is purely heuristic and assume that a bad parameter $\mathbf{p}$ (with respect to the objective function) is more likely to satisfy the quantified constraint system ξ.

After a box $[\mathbf{p}]$ is selected, it is split into several sub-boxes at line 5. Here we implement this procedure as a bisection with respect to the component of $[\mathbf{p}]$ with largest width. If the width of the box $[\mathbf{p}]$ is smaller than the precision $\epsilon_{\mathbf{p}}$, then it is not bisected and the *Split* procedure returns $[\mathbf{p}]$ itself. Then for each box $[\mathbf{p}]'$ obtained by splitting, several operations are applied. First, the box is contracted using contractors based on ψ, augmented with the constraint $v(\mathbf{p}) \leq \hat{v}$. This latter constraint enforces to discard parameters worse than $\hat{\mathbf{p}}$ from the search.

Next, the CSC procedure, described in Algorithm 2, is applied. Details on this procedure are given hereafter. This procedure returns a parameter $\tilde{\mathbf{p}}$ in $[\mathbf{p}]'$ and a status flag s. If this flag indicates *SAFE*, this means the parameter vector $\tilde{\mathbf{p}}$ satisfies $\xi(\tilde{\mathbf{p}}, \mathbf{x}), \forall \mathbf{x} \in [\mathbf{x}]$. If s indicates *UNSAFE*, a counter-example for the entire box $[\mathbf{p}]'$ has been found, i.e. $\exists \mathbf{x} \in [\mathbf{x}]$ such that $\neg\xi(\mathbf{p}, \mathbf{x})$ for all $\mathbf{p} \in [\mathbf{p}]'$. The flag indicates *UNKNOWN* if no conclusion on $[\mathbf{p}]'$ nor $\tilde{\mathbf{p}}$ could be made. If the flag is *SAFE*, the best viable parameter $\hat{\mathbf{p}}$ and its objective value $\hat{v}$ can be updated if it improves the current best one $\hat{\mathbf{p}}$. In addition, any box $[\mathbf{p}]$ in $\mathcal{L}$ such that $\min{[v]}([\mathbf{p}]) > \hat{v}$ are removed from the heap. Otherwise if the flag is *UNKNOWN* and the box $[\mathbf{p}]'$ has reached the prescribed precision $\epsilon_{\mathbf{p}}$, it is inserted back into the heap $\mathcal{L}$. *UNSAFE* boxes are simply discarded from further research. Finally, a stopping criteria is evaluated at line 17: the Best-FPS loop stops once the heap is empty (no more parameters to proceed) or another stopping criteria (e.g. maximal time, maximum number of iterations, etc) is satisfied. The algorithm returns then the best parameter vector $\hat{\mathbf{p}}$ found, if any.

Algorithm 1: Best Feasible Point Searcher

```
Input: Parameter box $\mathcal{P}$, State box $[\mathbf{x}]_{\text{init}}$ enclosing $K$,
       constraint system $\xi$ and $\psi$, objective $v$ and
       precision $\epsilon_{\mathbf{p}}$
Output: Parameter vector $\hat{\mathbf{p}}$
1  $\mathcal{L} \leftarrow \{\mathcal{P}\}$ ;                    /* Heap structure */
2  $\hat{v} \leftarrow \infty$; $\hat{\mathbf{p}} \leftarrow$ null;
3  while $\mathcal{L} \neq \emptyset$ and $\neg$stop do
4      $[\mathbf{p}] \leftarrow$ Extract($\mathcal{L}$);
5      $\mathcal{S} \leftarrow$ Split($[\mathbf{p}], \epsilon_{\mathbf{p}}$);
6      for $[\mathbf{p}]' \in \mathcal{S}$ do
7          $[\mathbf{p}]' \leftarrow$ Contract($\psi \wedge (v(\mathbf{p}) \leq \hat{\mathbf{p}}), [\mathbf{p}]'$);
8          if $[\mathbf{p}]' = \emptyset$ then continue;
9          $(\tilde{\mathbf{p}}, s) \leftarrow$ CSC($[\mathbf{p}]', [\mathbf{x}]_{\text{init}}, \xi$);
10         if $s$ is Safe and $\hat{v} > v(\tilde{\mathbf{p}})$ then
11             Update($\hat{v}, \hat{\mathbf{p}}, \mathcal{L}$);
12         end
13         if $s$ is not Unsafe and wid$[\mathbf{p}]' > \epsilon_{\mathbf{p}}$ then
14             Insert($\mathcal{L}, [\mathbf{p}]'$);
15         end
16     end
17     stop $\leftarrow$ CheckStoppingCriteria();
18 end
19 return $\hat{\mathbf{p}}$;
```

The CSC procedure, described in Algorithm 2, is a reformulation of the original one in [8]. The algorithm iterates over a stack of boxes in the universally quantified space $\mathbf{x}$, initially containing the whole box $[\mathbf{x}]_{\text{init}}$ at line 1. The goal is, though a decomposition of the box $[\mathbf{x}]_{\text{init}}$, to either validate a parameter vector $\tilde{\mathbf{p}}$ - selected at line 2- for all the boxes of the stack, or find one box $[\mathbf{x}]$ containing a vector $\mathbf{x}$ invalidating all the parameter box $[\mathbf{p}]$. By using a stack, a depth first search is performed. This choice of strategy is motivated by the fact that if $\tilde{p}$ is a valid parameter vector, there is no particular search strategy that will be better than another for proving its validity as all the boxes from the decomposition must be checked. However, if the parameter box $[\mathbf{p}]$ is not valid, finding quickly the corresponding vector $\mathbf{x}$ invalidating it is important. The depth-first search is suited for this task.

Each box $[\mathbf{x}]$ taken from the stack at the beginning of each loop are subject to different operations. First, a fast check of consistency of $[\mathbf{x}]$ is performed at line 5. A contractor on ξ is applied to the whole box $([\mathbf{p}], [\mathbf{x}])$ and if $[\mathbf{x}]$ is indeed contracted, then there are some of the $\mathbf{x}$ in $[\mathbf{x}]$ that do not satisfy $\xi(\mathbf{p}, \mathbf{x})$ for any $\mathbf{p} \in [\mathbf{p}]$. The parameter box is then not valid and an *UNSAFE* flag is returned. Then, the validity of the box $[\mathbf{x}]$ with respect to $\tilde{\mathbf{p}}$ and ξ is checked at line 7. A safe over-approximation of the evaluation of the constraint system ξ on $(\tilde{\mathbf{p}}, [\mathbf{x}])$ is performed by means of interval valuations with interval extensions. Next, the box $[\mathbf{x}]$ is contracted at line 10 using a contractor on $\neg\xi$ applied to $([\mathbf{p}], [\mathbf{x}])$. Contrary to the first contractor, here each $\mathbf{x} \in [\mathbf{x}]$ effectively removed from the contracted box $[\mathbf{x}]''$ are satisfying ξ for all parameters in $[\mathbf{p}]$, validating them. Afterwards, a *Disprove* procedure with respect to ξ is applied to $[\mathbf{p}]$ and $[\mathbf{x}]'$. This procedure is equivalent to the proof procedure of [12] on the constraint system $\neg\xi$. Starting from $([\mathbf{p}], \text{mid}\,[\mathbf{x}])$, a parametric interval Newton operator -an Hansen-Sengupta operator- with inflation is applied on each system of equations involved in each conjunction of constraints of $\neg\xi$ described by (12). The elements of $[\mathbf{p}]$ are selected as parameters for the parametric interval Newton operator, hence performing an attempt to prove the existence of some $\mathbf{x}$ invalidating all $[\mathbf{p}]$. If a certified enclosure $[\mathbf{x}_\mathcal{N}]$ of the solutions to the system of equation is found (e.g. $\forall \mathbf{p} \in [\mathbf{p}], \ \exists \mathbf{x} \in [\mathbf{x}_\mathcal{N}], \ h(\mathbf{p}, \mathbf{x}) = 0)$, then the remaining inequalities are checked on $([\mathbf{p}], [\mathbf{x}_\mathcal{N}])$ by means of interval evaluations. If this check is successful, the box $[\mathbf{x}_\mathcal{N}]$ is proved to contain a $\mathbf{x}$ satisfying $\neg\xi(\mathbf{p}, \mathbf{x})$ for all $\mathbf{p} \in [\mathbf{p}]$. The parameter box $[\mathbf{p}]$ is hence not valid.

Eventually, if none of those previous steps could validate nor invalidate the current iterated box, then it is bisected if it is has not reached the prescribed precision $\epsilon_\mathbf{x}$. The two new boxes are pushed back into the stack. If the iterated box is indeed too small, then CSC stops by returning an *UNKNOWN* flag. We could still continue to explore the space of $\mathbf{x}$ in order to invalidate $[\mathbf{p}]$, but it is not possible to validate $\tilde{\mathbf{p}}$ anymore. In the end, if the loop is exited after validating all the boxes from the stack, then a *SAFE* flag is returned as well as the valid parameter vector $\tilde{\mathbf{p}}$.

Complexity. We can note that the Best-FPS-CSC algorithm is a decomposition algorithm in the parameter space which nests another decomposition algorithm in the universally quantified state space. This entails an exponential time algorithm in the joint parameter and state space. With such a complexity, the algorithm is limited to relatively small problems, both in terms of number of states and parameters of the template. We can note however that a complete exploration of the parameter space is not necessary. Hence, using stopping conditions and search heuristics in Algorithm 1 that can lead to quickly find a sufficiently good valid parameter $\hat{\mathbf{p}}$ might ease the computational burden.

3.2 Improvement of the results

From Proposition 3.1, when a function h is found, the set $\mathcal{H}$ can easily be inner-approximated using interval analysis in a branch and prune algorithm. It provides a first inner approximation through a paving with boxes. Proposition 2.1 is then used to design an iterative scheme that allows to improve this inner approximation. For a given box $[\tilde{\mathbf{x}}]$, if we find an input $\mathbf{u}$ such that for some $\tilde{t} \in [0, t_\text{end}]$, the state $\mathbf{x}(\tilde{t})$ is included in the inner-approximation of the viability kernel we have produced so far, and such that for all $t \in \left[0, \tilde{t}\right]$, $x(t) \in K$ then $[\tilde{\mathbf{x}}]$ belongs to $\text{Viab}_S(K)$. This property can be used in an iterative approach to improve the results of the first inner-approximation.

At each iteration we consider the boxes of the paving belonging to the direct neighborhood of the current inner-approximation and test this before-mentioned property. To find this input $\mathbf{u}$, we sample $\mathcal{U}$ to a finite list of inputs and use each of them with the box $[\tilde{\mathbf{x}}]$. If the property is verified, the box is then added to the current inner approximation of the viability kernel. If not, and if a given precision is not reached for the box, it is bisected. If in an iteration, no box is added to the inner approximation or bisected, the algorithm stops.

Improvements were considered for this iteration scheme in the implementation used in the next section compared to [20]. For example, when an input $\mathbf{u}$ has allowed to prove a box to belong to $\text{Viab}_S(K)$, it will be the first one to be tested

Algorithm 2: Computable Sufficient Condition

```
Input: Parameter box $[\mathbf{p}]$, State box $[\mathbf{x}]_\text{init}$, constraint
       system $\xi$ and precision $\epsilon_x$
Output: Vector $\tilde{\mathbf{p}}$ and a status flag $s$
1  $\mathcal{S} \leftarrow \{[\mathbf{x}]_\text{init}\}$ ;              /* Stack structure */
2  $\tilde{\mathbf{p}} \leftarrow$ Select($[\mathbf{p}]$) ;            /* E.g. $\tilde{\mathbf{p}} = \text{mid}([\mathbf{p}])$ */
3  while $\mathcal{S} \neq \emptyset$ do
4      $[\mathbf{x}] \leftarrow$ Pop($\mathcal{S}$);
5      $([\mathbf{p}]', [\mathbf{x}]') \leftarrow$ Contract($\xi, [\mathbf{p}], [\mathbf{x}]$);
6      if $[\mathbf{x}]' \neq [\mathbf{x}]$ then return ($\tilde{\mathbf{p}}$, Unsafe) ;
7      if $\xi(\tilde{p}, [\mathbf{x}])$ then
8          continue ;          /* Validation of $[\mathbf{x}]$ */
9      end
10     $([\mathbf{p}]'', [\mathbf{x}]'') \leftarrow$ Contract($\neg\xi, [\mathbf{p}], [\mathbf{x}]$);
11     if Disprove$_\xi$($[\mathbf{p}], [\mathbf{x}]''$) then
12         return ($\tilde{\mathbf{p}}$, Unsafe);
13     end
14     if wid$[\mathbf{x}]'' \leq \epsilon_\mathbf{x}$ then
15         return ($\tilde{\mathbf{p}}$, Unknown)
16     else
17         $([\mathbf{x}]_1, [\mathbf{x}]_2) \leftarrow$ Bisect($[\mathbf{x}]''$);
18         Push($\mathcal{S}, [\mathbf{x}]_1$); Push($\mathcal{S}, [\mathbf{x}]_2$);
19     end
20 end
21 return ($\tilde{\mathbf{p}}$, Safe);
```

for the next non-determined boxes neighboring $[\tilde{\mathbf{x}}]$. Also at each iteration, each epsilon box (boxes that have reached the prescribed precision) has to be checked again since it can reach a box newly added to $\mathrm{Viab}_S(K)$ in the previous iteration. The results from validated numerical integration have then to be stored for all epsilon box to not compute it again.

As one can see, this improvement phase is an expensive procedure as many validated numerical integration processes have to be repeated on a paving of the state space. This justify the need of computing an inner-approximation of the viability kernel in the first-phase which is as wide as possible, in a reasonable amount of time.

4 RESULTS

We implemented the two phases of the algorithm in C++ using Ibex and DynIbex [2], a plugin of Ibex providing rigorous numerical integration tools using Runge-Kutta methods [1] for the improvement phase [3]. All the experiments have been ran on Linux Ubuntu 16.04, with CPU Intel i7 2.5 Ghz and 16 Go RAM.

4.1 Car on the hill

We compute the viability kernel on the "car on the hill" problem whose system is described as follows:

$$\begin{cases} \dot{y}_1 = y_2(t) \\ \dot{y}_2 = -z(y1(t), y2(t)) + u(t) \end{cases}, \tag{13}$$

where

$$z(y_1, y_2) = 9.81 \sin\left(\frac{1.1 \sin(1.2y_1) + 1.2 \sin(1.1y_1)}{2}\right) + 0.7y_2.$$

This system for a given constant control u has several equilibrium points at $y_2 = 0$. We will use the equilibrium points with $u = 0$ for initializing template functions. Here there are 5 such points.

First, we consider the safe set to be the box $K = ([-1, 13] \times [-6, 6])^T$. The control function u is bounded in $[-3, 3]$.

We consider the following 3-parametric template function:

$$h_{\mathbf{p}}(\mathbf{y}) := p_1(y_1 - c)^2 + 2p_2y_2^2 + p_3(y_1 - c)y_2 - 1, \tag{14}$$

i.e. a quadratic form where we additionally impose a positive definite constraint on the parameters (i.e. $h_{\mathbf{p}}$ must be convex). This quadratic form is centered on an equilibrium point $(c, 0)$. The set of control functions are sampled to the following ones: the constant functions $u_1 := -3$, $u_2 := 0$ and $u_3 := 3$ in addition to the (bounded) feedback control

$$u_4 := \min(3, \max(-3, \tilde{u}_4)),$$
$$\tilde{u}_4 := \frac{1}{2}(c - y_1(t)) - \frac{3}{2}y_2(t) + z(y_1(t), y_2(t))$$

which has been obtained by linearizing feedback. As an objective function to optimize, we simply chose to minimize the square of the 2-norm of (p_1, p_2) whose minimization approximates the maximization of the volume of the region defined by $h_{\mathbf{p}}(\mathbf{x}) \leq 0$. The parameter domain is $[0.025, 10] \times [-5, 5] \times [0.025, 10]$, with $\epsilon_{\mathbf{p}} = 0.025$. For the CSC procedure, $\epsilon_{\mathbf{x}}$ is set to $\max(0.00125, \mathrm{wid}[\mathbf{p}]/2)$, with $[\mathbf{p}]$ the input parameter box. Note eventually that we fixed a timeout to 10 seconds for each template function to find.

We compare our initial phase of the method with the one from [20] (the improvement phase are identical). Results for the initial inner-approximation of the viability kernel are shown on Figure 2.

It took 30 seconds for our method to produce the union of ellipsoids shown on the top of Figure 2, while it took about 5 seconds to obtain the ones with the original method. It

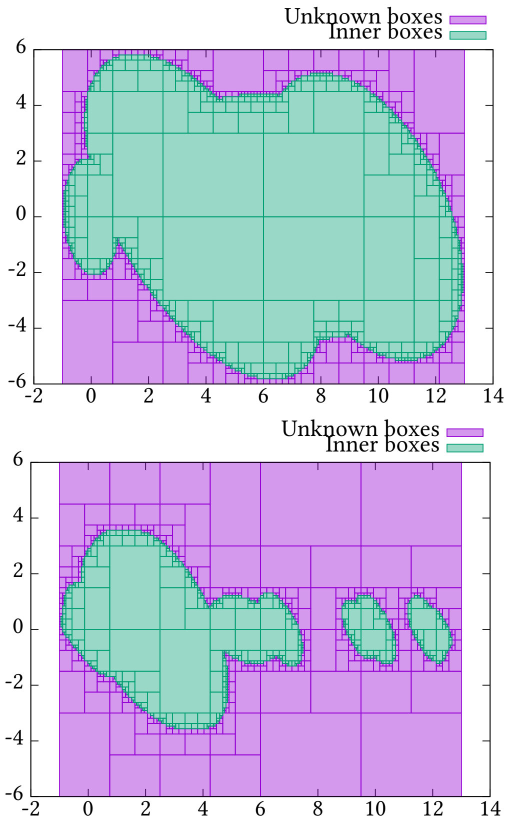

Figure 2: (Car on the hill) Computed initial inner-approximation of $\mathrm{Viab}_S(K)$: with our new method (top); with the original method (bottom).

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

[3]Sources and a virtual machine for running the experiments are available at http://ben-martin.fr/hscc-2018-sources/

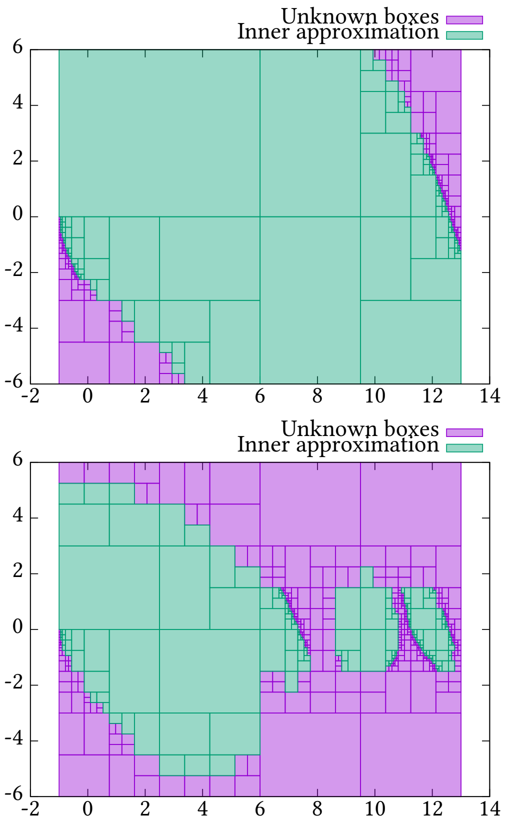

Figure 3: (Car on the hill) Improvement method after 5 iterations: starting with our new method (top); starting with the original method (bottom).

is clear that our approach produces a larger initial inner-approximation. The extra computations required to compute it trades-off with the required computations for the improvement phase. Figure 3 shows the result of the improvement procedure after 5 iterations.

After 5 iterations, the current inner-approximation of the viability kernel is wider if one uses our presented approach than with the original one. In the end, it takes overall 150 seconds for our approach to converge to an accurate inner-approximation of $\mathrm{Viab}_S(K)$, while it takes 300 seconds in the original approach. This final inner-approximation is depicted on Figure 4.

We consider now that $K := ([-1, 13] \times [-6, 6])^T \setminus (([4, 8] \times [2, 6]) \cup ([4, 8] \times [-6, -2]))^T$. We show here how the method behave when the set K is not convex. Figure 5 shows the initial ellipsoids obtained by our method, the results after 5 iterations of the improvement procedure and the final inner-appoximation obtained. It took overall 400 seconds to obtain this result.

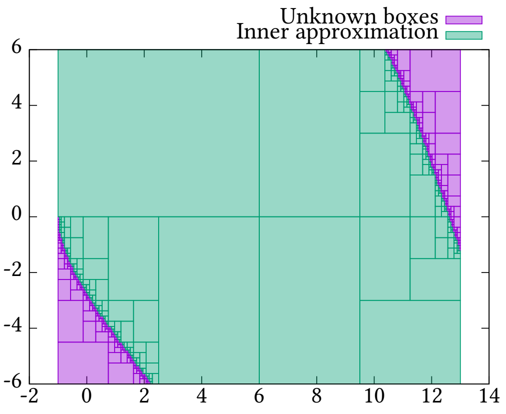

Figure 4: (Car on the hill) Improvement method at the end of the computation with our new method.

The inner-approximation produced by the first phase via the presented Best-FPS-CSC is well suited for such complex safe sets compared to the previous method. In [20], the validity of the ellipses is only checked regarding their frontier. It will then fail to detect "holes" within K, forbidden areas that can occur inside a considered ellipse.

4.2 Balancing an heavy object on a light rod

A second experiment has been conducted, this time with an input of dimension 2. It demonstrates another improvement using the method described in Section 3.1 compared to [20] since it fails to produce ellipses for dimension of the input greater than one. We used the equations of motion of a point mass balancing on a rod:

$$\begin{cases} \dot{y}_1 = y_2(t) \\ \dot{y}_2 = \cos(y_1(t))\,(u_1(t)y_1(t) + u_2(t)y_2(t)) + 9.81\sin(y_1(t)). \end{cases} \quad (15)$$

using $K = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [-\pi, \pi]$ and $\mathcal{U} = [-15, 15] \times [-2, 2]$. The same template and same objective function as the one for the previous experiment is considered here, but with the only center $(0, 0)$. We consider a sample of constant control functions $(u_1, u_2) \in \{-15, 0, 15\} \times \{-2, 0, 2\}$. We fixed a timeout of 30 seconds for the first phase. Figure 6 provides the results of the first phase of our method.

The final result is shown in Figure 7. It tooks overall 680 seconds of computations to obtain this result. A larger number of inputs increase both the size of the constraint system (11) solved in the first phase and the sampling of control during the second phase. Overall, this does not impact much the computation process as the system is small.

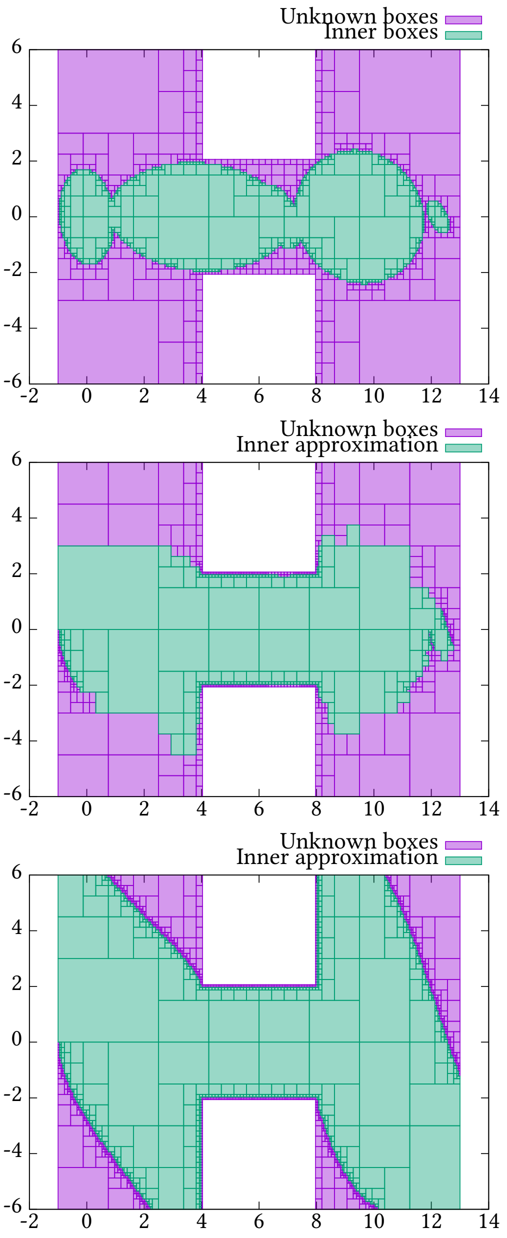

Figure 5: (Car on the hill) Non-convex K: initial inner-approximation (top); after 5 iterations of the improvement phase (middle); final inner-approximation (bottom).

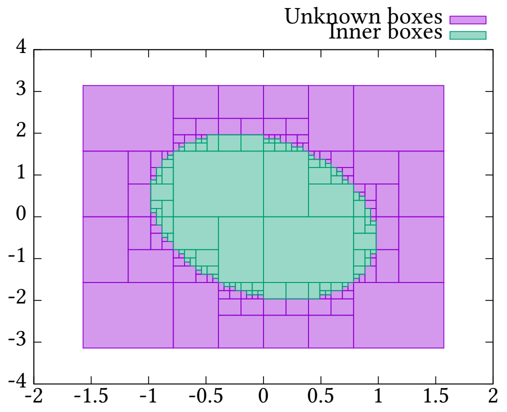

Figure 6: (Balancing) First inner-approximation.

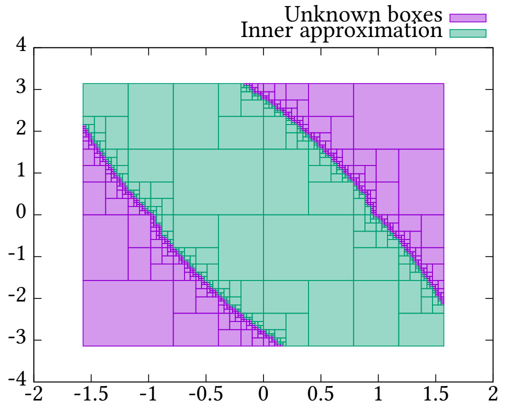

Figure 7: (Balancing) Improvement method at the end of the computation with our new method.

5 CONCLUSION

We have presented in this paper a rigorous method based on interval analysis for computing viability kernels following the framework proposed in [20]. In particular we investigated another approach for generating initial inner-approximation of the viability kernel based on the computation of specific parametric template function. This method is inspired by the method from [8], for the computation of barrier functions. We propose some additional elements to this method, namely extra tests for rejecting parameters not leading to a valid function entailing an inner-approximation, and an optimization loop to compute the set of parameters for this function leading to a larger induced inner-approximation of the viability kernel. A comparison of the new method with the original one is presented, showing better performances compared to the original approach.

There are many further research directions for this method, the next biggest issue to tackle being its scalability. To this end, we need to study more efficient search strategies, in particular with a better focus on quickly finding valid parameter for the template functions. Having a measure of validity of parameters within a given box may help to direct the search towards more promising regions of the parameter space. We can also consider the adaptation of the method from [11] to our problem, i.e. having a method that quickly finds a valid parameter vector within a parameter box, avoiding an expensive decomposition of the state space at each iteration of the algorithm. Last, we can think about using another optimization scheme, a heuristic one without a complete search within the parameter space. Such a method would be possible provided an approximate model of the valid parameters, and of the objective function if necessary, can be computed in order to guide the heuristic search. Eventually, a complete study of the merits and drawbacks of our (improved) method compared to other techniques from the literature, such as [16], needs to be made.

ACKNOWLEDGMENTS

We would like to thanks the referees for the useful comments. This work is supported by the DGA/MRIS project "Sureté de Fonctionnement des Systèmes Robotiques Complexes".

REFERENCES

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

[2] Julien Alexandre dit Sandretto and Alexandre Chapoutot. 2016. Validated simulation of differential algebraic equations with Runge-Kutta methods. *Reliable Computing* 22, pp. 56–77 (July 2016).

[3] Jean-Pierre Aubin. 2009. *Viability theory.* Springer Science & Business Media. https://doi.org/10.1007/978-0-8176-4910-4

[4] Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. 2013. Enclosing Temporal Evolution of Dynamical Systems Using Numerical Methods. In *NASA Formal Methods (LNCS).* Springer, 108–123. https://doi.org/10.1007/978-3-642-38088-4_8

[5] Olivier Bouissou and Matthieu Martel. 2006. GRKLib: a Guaranteed Runge Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics.* https://doi.org/10.1109/SCAN.2006.20

[6] Gilles Chabert and Luc Jaulin. 2009. Contractor programming. *Artificial Intelligence* 173, 11 (2009), 1079 – 1100. https://doi.org/10.1016/j.artint.2009.03.002

[7] Guillaume Deffuant, Laetitia Chapel, and Sophie Martin. 2007. Approximating Viability Kernels With Support Vector Machines. *IEEE Trans. Automat. Control* 52, 5 (May 2007), 933–937. https://doi.org/10.1109/TAC.2007.895881

[8] Adel Djaballah, Alexandre Chapoutot, Michel Kieffer, and Olivier Bouissou. 2017. Construction of parametric barrier functions for dynamical systems using interval analysis. *Automatica* 78 (2017), 287 – 296. https://doi.org/10.1016/j.automatica.2016.12.013

[9] Karol Gajda, Andrzej Marciniak, and Barbara Szyszka. 2000. Three- and Four-Stage Implicit Interval Methods of Runge-Kutta Type. *Computational Methods in Science and Technology* 6, 1 (2000), 41–59. https://doi.org/10.12921/cmst.2000.06.01.41-59

[10] Antoine Girard. 2012. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica* 48, 5 (2012), 947 – 953. https://doi.org/10.1016/j.automatica.2012.02.037

[11] Milan Hladík and Stefan Ratschan. 2014. Efficient Solution of a Class of Quantified Constraints with Quantifier Prefix Exists-Forall. *Mathematics in Computer Science* 8, 3 (2014), 329–340. https://doi.org/10.1007/s11786-014-0195-8

[12] Daisuke Ishii, Alexandre Goldsztejn, and Christophe Jermann. 2012. Interval-based projection method for under-constrained numerical systems. *Constraints* 17, 4 (2012), 432–460. https://doi.org/10.1007/s10601-012-9126-y

[13] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. 2001. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics.* Springer-Verlag. https://doi.org/10.1007/978-1-4471-0249-6

[14] Shahab Kaynama, John Maidens, Meeko Oishi, Ian M. Mitchell, and Guy A. Dumont. 2012. Computing the Viability Kernel Using Maximal Reachable Sets. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '12).* ACM, New York, NY, USA, 55–64. https://doi.org/10.1145/2185632.2185644

[15] R. Baker Kearfott. 1996. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers. https://doi.org/10.1007/978-1-4757-2495-0

[16] Milan Korda, Didier Henrion, and Colin N. Jones. 2013. Convex computation of the maximum controlled invariant set for discrete-time polynomial control systems. In *52nd IEEE Conference on Decision and Control.* 7107–7112. https://doi.org/10.1109/CDC.2013.6761016

[17] Andrzej Marciniak. 2004. Implicit Interval Methods for Solving the Initial Value Problem. *Numerical Algorithms* 37, 1-4 (2004), 241–251. https://doi.org/10.1023/B:NUMA.0000049471.81341.60

[18] Andrzej Marciniak and Barbara Szyszka. 2004. On Representations of Coefficients in Implicit Interval Methods of Runge-Kutta Type. *Computational Methods in Science and Technology* 10, 1 (2004), 57–71. https://doi.org/10.12921/cmst.2004.10.01.57-71

[19] Dominique Monnet, Luc Jaulin, Jordan Ninin, Alexandre Chapoutot, and Julien Alexandre-dit Sandretto. 2015. Viability kernel computation based on interval methods. In *SWIM (Summer Workshop on Interval Analysis).*

[20] D. Monnet, J. Ninin, and Luc Jaulin. 2016. Computing an Inner and an Outer Approximation of the Viability Kernel. *Reliable Computing* 22, 1 (Sep 2016), 138–148.

[21] Ramon E. Moore. 1966. *Interval Analysis.* Prentice-Hall.

[22] Bouguerra Muhammad, Thierry Fraichard, and Mohamed Fezari. 2015. Safe Motion using Viability Kernels. In *ICRA 2015-IEEE Int. Conf. on Robotics and Automation.* 3259–3264. https://doi.org/10.1109/ICRA.2015.7139648

[23] Nedialko S Nedialkov, Kenneth R Jackson, and George F Corliss. 1999. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. Comput.* 105, 1 (1999), 21–68. https://doi.org/10.1016/S0096-3003(98)10083-8

[24] Arnold Neumaier. 1991. *Interval Methods for Systems of Equations.* Cambridge University Press. https://doi.org/10.1017/CBO9780511526473

[25] Bertrand Neveu, Gilles Trombettoni, and Ignacio Araya. 2016. Node selection strategies in interval Branch and Bound algorithms. *Journal of Global Optimization* 64, 2 (2016), 289–304. https://doi.org/10.1007/s10898-015-0375-3

[26] Gunther Reissig, Alexander Weber, and Matthias Rungger. 2017. Feedback Refinement Relations for the Synthesis of Symbolic Controllers. *IEEE Trans. Automat. Control* 62, 4 (April 2017), 1781–1796. https://doi.org/10.1109/TAC.2016.2593947

[27] Patrick Saint-Pierre. 1994. Approximation of the viability kernel. *Applied Mathematics and Optimization* 29, 2 (01 Mar 1994), 187–209. https://doi.org/10.1007/BF01204182

[28] Zhikun She and Bai Xue. 2013. Computing an invariance kernel with target by computing Lyapunov-like functions. *IET Control Theory & Applications* 7, 15 (2013), 1932–1940. https://doi.org/10.1049/iet-cta.2013.0275