Set-membership Method for Discrete Optimal Control

Rémy Guyonneau, Sébastien Lagrange, Laurent Hardouin, Mehdi Lhommeau
Laboratoire d'Ingénierie des Systèmes Automatisés (LISA), Université d'Angers,
62 Avenue Notre Dame du Lac, Angers, France

Keywords: Non-linear Systems, Interval Analysis, Guaranteed Numerical Integration, Optimal Control.

Abstract: The objective of this paper is twofold. First we propose a new approach for computing $\mathbf{C}_{t_0,t_f}$ the subset of initial states of a system from which there exists at least one trajectory reaching a target $\mathbf{T}$ in a finite time t_f from a time t_0. This is done considering a discrete time t_k and a control vector continuous over a time $[t_{k-1},t_k]$. Then, using the previously mentioned work and given a cost function, the objective is to estimate an enclosure of the discrete optimal control vector from an initial state of $\mathbf{C}_{t_0,t_f}$ to the target. Whereas classical methods do not provide any guaranty on the set of state vectors that belong to the $\mathbf{C}_{t_0,t_f}$, interval analysis and guaranteed numerical integration allow us to avoid any indetermination. We present an algorithm able to provide guaranteed characterizations of the inner $\mathbf{C}^-_{t_0,t_f}$ and an the outer $\mathbf{C}^+_{t_0,t_f}$ of $\mathbf{C}_{t_0,t_f}$, such that $\mathbf{C}^-_{t_0,t_f} \subseteq \mathbf{C}_{t_0,t_f} \subseteq \mathbf{C}^+_{t_0,t_f}$. In addition to that, the presented algorithm is extended in order enclose the discrete optimal control vector of the system, form an initial state to the target, by a set of discrete trajectories.

1 INTRODUCTION

We consider a control system, defined by the differential equation

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

where $\mathbf{x}(t) \in \mathbb{R}^n$ be the state vector of the system, $\mathbf{u}(t) \in \mathbf{U}$ be the control vector. This system is studied over a bounded time $t_k \in [t_0,t_f]$, considering a discrete time

$$t_k = t_0 + k \times \delta_t, t_k \leq t_f, k \in \{1,\cdots,m\}, \tag{2}$$

It will be assumed that δ_t is small enough so that the control vector $\mathbf{u}(t)$ can be assumed to be continuous over $[t_k,t_{k+1}]$. Associated to the differential equation (1) we define the *flow map*

$$\varphi(t_0,t_k;\mathbf{x}_0,\mathbf{u}(t)) = \mathbf{x}(t), \tag{3}$$

where $\mathbf{x}(t)$ denotes the solution to (1) with the initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$ and the control function $\mathbf{u}(t) \in \mathcal{U}$, where $\mathcal{U} = \{\mathbf{u} : [t_0,t_{k-1}] \to \mathbf{U} | \mathbf{u}$ is continuous over $[t_k,t_{k+1}]\}$ denotes the set of admissible controls. Note that in the later the notation $\mathbf{u}_k$ will refer to $\mathbf{u}(t) : [t_k,t_{k+1}] \to \mathbf{U}$, with $\mathbf{u}(t)$ continuous over $[t_k,t_{k+1}]$. Given $\mathbf{X}_0$ a set of possible initial values $\mathbf{x}_0$, the reachable set of the system (1) at the time t_k is

$$\varphi(t_0,t_k;\mathbf{X}_0,\mathbf{U}) = \{ \begin{array}{l} \varphi(t_0,t_k;\mathbf{x}_0,\mathbf{u}(t)) | \\ \varphi(t_0,t_0;\mathbf{x}_0,\mathbf{u}(t)) = \mathbf{x}_0 \\ \text{and } \varphi : [t_0,t_k] \times \mathbf{X}_0 \times \mathbf{U} \to \mathbb{R}^n \\ \text{is a solution of (1) for some} \\ \mathbf{u}(t) \in \mathcal{U}\}. \end{array} \tag{4}$$

The trajectory from $\underline{t}_k$ to $\overline{t}_k$ is defined by

$$\phi([\underline{t}_k,\overline{t}_k];\mathbf{X},\mathbf{U}) = \{ \begin{array}{l} \tilde{\mathbf{X}} \subseteq \mathbb{R}^n | \exists t_k \in [\underline{t}_k,\overline{t}_k], \\ \tilde{\mathbf{X}} = \varphi(\underline{t}_k,t_k;\mathbf{X},\mathbf{U})\}. \end{array} \tag{5}$$

Let $\mathbf{K} \subset \mathbb{R}^n$ be a state constraint such that $\mathbf{x}(t) \in \mathbf{K}$, and $\mathbf{T}$ be a compact set in $\mathbf{K}$ (the target). $\mathbf{C}_{t_0,t_f}$ corresponds to the subset of initial states of $\mathbf{K}$ from which there exists at least one solution of (1) reaching the target $\mathbf{T}$ in finite time t_f from a time t_0:

$$\mathbf{C}_{t_0,t_f} = \{\mathbf{x}_0 \in \mathbf{K} | \exists \mathbf{u}(t) \in \mathcal{U}, \varphi(t_0,t_f;\mathbf{x}_0,\mathbf{u}(t)) \in \mathbf{T}\} \tag{6}$$

Given that the input vector is continuous over $[t_k,t_{k+1}]$, the first objective of this paper is to compute an inner and outer approximations, $\mathbf{C}^-_{t_0,t_f}$ and $\mathbf{C}^+_{t_0,t_f}$, of $\mathbf{C}_{t_0,t_f}$ (Lhommeau et al., 2011; Delanoue et al., 2009). Such problems of dynamics control under constraints refer to viability theory (Aubin, 2006) (see (Aubin, 1990) for a survey). The proposed method to characterize $\mathbf{C}_{t_0,t_f}$, which has similarities with dynamic programming (Kirk, 2004), has the advantage that it is guaranteed whereas numerical methods give only an approximation. An interval analysis based method

Guyonneau R., Lagrange S., Hardouin L. and Lhommeau M..
Set-membership Method for Discrete Optimal Control.
DOI: 10.5220/0004458001930200
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 193-200
ISBN: 978-989-8565-70-9

is used to compute the approximations, such that $\mathbf{C}^-_{t_0,t_f} \subseteq \mathbf{C}_{t_0,t_f} \subseteq \mathbf{C}^+_{t_0,t_f}$, by using guaranteed numerical integration (VNODE-LP[1]). Note that an obvious approximations would be $\mathbf{C}^-_{t_0,t_f} = \emptyset$ and $\mathbf{C}^+_{t_0,t_f} = \mathbf{K}$. The proposed method aims at computing a better enclosure of $\mathbf{C}_{t_0,t_f}$.

In a second part, we got interested to the optimal control problem. Given a cost function J and an initial state $\mathbf{x}_0 \in \mathbf{C}_{t_0,t_f}$, we propose a numerical method to evaluate an enclosure of the discrete optimal control $\mathbf{u}(t) \in \mathbf{U}$ such that $\varphi(t_0,t_f;\mathbf{x}_0,\mathbf{u}(t)) \in \mathbf{T}$ and $\mathbf{u}(t)$ continuous over $[t_k,t_{k+1}]$.

The paper is organised as follow. First some interval analysis tools are presented in Section 2 as they are used to compute the inner and outer approximations. Section 3 presents the proposed algorithm to compute $\mathbf{C}_{t_0,t_f}$ and is followed by experimental results in Section 4. Finally Section 5 discusses about the optimal control problem and Section 6 concludes this paper.

2 INTERVAL ANALYSIS

Interval analysis for ordinary differential equations was introduced by Moore (Moore, 1966) (See (Nedialkov et al., 1999) for a description and bibliography on this topic). These methods provide numerically reliable enclosures of the exact solution of differential equations.

Interval analysis usually considers only closed intervals. The set of these intervals is denoted $\mathbb{IR}$. An interval is usually denoted using brackets. An element of an interval $[x]$ is denoted by x. An interval vector (box) $[\mathbf{x}]$ of $\mathbb{R}^n$ is a Cartesian product of n intervals. If $[\mathbf{x}] = [\underline{x}_1,\overline{x}_1] \times \cdots \times [\underline{x}_n,\overline{x}_n]$ is a box, then its width is

$$w([\mathbf{x}]) = w([x_1]) \times \cdots \times w([x_n]), \tag{7}$$

where $w([x_i]) = \overline{x}_i - \underline{x}_i$. The set of all boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$.

The *Bisect*() function divides an interval $[x]$ into two intervals $[x_1]$ and $[x_2]$ such as $[x_1] \cup [x_2] = [x]$, $[x_1] \cap [x_2] = \emptyset$ and $w([x_1]) = w([x_2])$.

The main concept of interval analysis is the extension of real functions to intervals, which is defined as follows. Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a continuous real function, and $[\mathbf{f}] : \mathbb{IR}^n \to \mathbb{IR}^m$ be an inclusion function. Then $[\mathbf{f}]$ is an inclusion function of $\mathbf{f}$ if and only if for every $[\mathbf{x}] \in \mathbb{IR}^n, \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\} \subseteq [\mathbf{f}]([\mathbf{x}])$.

Hence, an interval inclusion allows computing enclosures of the image of boxes by real functions. It now remains to show how to compute such inclusions. The first step is to compute formally the interval extension of elementary functions. For example, we define $[\underline{x},\overline{x}] + [\underline{y},\overline{y}] := [\underline{x}+\underline{y},\overline{x}+\overline{y}]$. Similar simple expressions are obtained for other functions like $-, \times, \div, x^n, \sqrt{x}, exp, \cdots$ This process gives rise to the so-called *interval arithmetic* (see (Jaulin et al., 2001)).

Then, an interval inclusion for real functions compound of these elementary operations is simply obtained by changing the real operations to their interval counterparts. This interval inclusion is called the *natural extension*.

Interval arithmetic can be used to compute guaranteed integration. In the later, the Nedialkov method is used to compute:

- $[\mathbf{x}]^*$ such that $[\mathbf{x}]^* \supset \varphi(t_k,t_{k+1};[\mathbf{x}],[\mathbf{u}_k])$,
- $\mathbf{K}^*$ such that $\mathbf{K}^* \supset \phi([t_k,t_{k+1}];[\mathbf{x}],[\mathbf{u}_k])$.

Note that the Nedialkov method is one chosen solution over several methods, one could chose a different approach.

Given a bounded set $\mathbf{E}$ of complex shape, one usually defines an axis-aligned box or paving, i.e. an union of non-overlapping boxes, $\mathbf{E}^+$ which contains the set $\mathbf{E}$: this is known as the outer approximation of it. Likewise, one also defines an inner approximation $\mathbf{E}^-$ which is contained in the set $\mathbf{E}$. Hence we have the following property

$$\mathbf{E}^- \subseteq \mathbf{E} \subseteq \mathbf{E}^+ \tag{8}$$

3 CHARACTERIZATION OF $\mathbf{C}_{T_0,T_F}$

This section presents an algorithm able to provide an inner and an outer approximation of $\mathbf{C}_{t_0,t_f}$ assuming that the input $\mathbf{u}(t_k)$ is continuous over $[t_k,t_{k+1}]$, and bounded so it is possible to determinate a box $[\mathbf{u}_k]$ such that $\mathbf{u}(t) \in [\mathbf{u}_k]$ over $[t_k,t_{k+1}]$. That is, the obtained results will be dependant of the time's step δ_t.

For each time t_k the algorithm computes a gridding of $\mathbf{K}$ (a *slice*), noted $S(t_k)$. The resolution of the gridding is $\delta_\mathbf{K} = (\delta_{x_1},\cdots,\delta_{x_i},\cdots,\delta_{x_n})$ where δ_{x_i} corresponds to the resolution of the i^{th} dimension of $\mathbf{K}$ (Figure 1).

A cell s_i of $S(t_k)$ can be

- *unreachable* if no state $\mathbf{x}$ in this cell allows the system to reach the target at time t_f, for all possible input vectors. The set of all the *unreachable* cells of $S(t_k)$ is noted $S^u(t_k)$

$$S^u(t_k) = \{ \quad s_i \in S(t_k) | \forall \mathbf{u}(t) \in \mathbf{U}, \\ \phi([t_k,t_f];s_i,\mathbf{u}(t)) \cap \mathbf{T} = \emptyset\} \tag{9}$$

- *reachable* if for all the states $\mathbf{x}$ of this cell it exists an input vector that allows the system to reach the

[1] A C++ package for computing bounds on solutions in Initial Value Problems for Ordinary Differential Equations, by N. Nedialkov.

target at time t_f with a trajectory entirely included in the state space domain $\mathbf{K}$. The set of all the *reachable* cells of $S(t_k)$ is noted $S^r(t_k)$.

$$S^r(t_k) = \{s_i \in S(t_k) | \exists \mathbf{u}(t) \in \mathbf{U}, \varphi(t_k, t_f; s_i, \mathbf{u}(t)) \subseteq \mathbf{T} \text{ and } \phi([t_k, t_f]; s_i, \mathbf{u}(t)) \subseteq \mathbf{K}\} \quad (10)$$

- *indeterminate* if it is neither *reachable* or *unreachable*. The set of all the *indeterminate* cells of $S(t_k)$ is noted $S^i(t_k)$.

$$S^i(t_k) = S(t_k) \setminus (S^r(t_k) \cup S^u(t_k)) \quad (11)$$

It can be noticed that

$$\mathbf{C}^-_{t_0,t_f} = S^r(t_0), \quad \mathbf{C}^+_{t_0,t_f} = S^r(t_0) \cup S^i(t_0). \quad (12)$$

The time's step δ_t of the input vector (step time during which one the input vector is continuous and bounded) and the resolutions $\delta_{x_i}, i = 1, \cdots, n$, are defined by the desired precision of the $\mathbf{C}^+_{t_0,t_f}$ and $\mathbf{C}^-_{t_0,t_f}$ characterizations. Note that $\mathbf{C}^-_{t_0,t_f}$ can be empty if the precision is too rough.

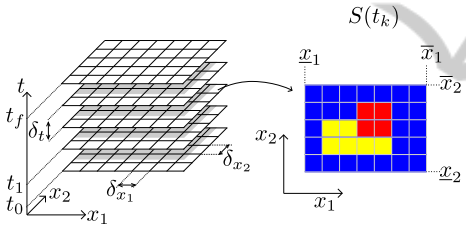

Figure 1: An example of slice set and slice $S(t_k)$, with $\mathbf{K} = ([\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}])$ a two dimensional state space domain. The following color scheme is held for all the figures of this paper: blue (dark grey) $\rightarrow$ *unreachable*, red (medium grey) $\rightarrow$ *reachable*, yellow (light grey) $\rightarrow$ *indeterminate*.

3.1 The $\mathbf{C}_{t_0,t_f}$ Algorithm

We propose an iterative algorithm (Algorithm 1) that computes for each slice $S(t_k)$, three subsets $S^r(t_k)$, $S^u(t_k)$ and $S^i(t_k)$ such as

$$\begin{aligned} S(t_k) &= S^r(t_k) \cup S^u(t_k) \cup S^i(t_k) \\ S^r(t_k) &= \{s_i \subseteq S(t_k) | s_i \text{ is } \textit{reachable}\} \\ S^u(t_k) &= \{s_i \subseteq S(t_k) | s_i \text{ is } \textit{unreachable}\} \\ S^i(t_k) &= \{s_i \subseteq S(t_k) | s_i \text{ is } \textit{indeterminate}\} \end{aligned} \quad (13)$$

Those subsets are computed from t_f to t_0 using guarantee numerical integration. After the initialization of the subsets at time t_f the other subsets at time t_k are built using the reachability information of the cells $s_i \subseteq S(t_{k+1})$. Note that the *ADD* algorithm is presented in Section 3.2.

Lines 1 to 3 of the Algorithm 1 initialise the three subsets of the slice $S(t_f)$. For this particular slice, the *reachable* cells are the ones included in the target, the

Algorithm 1: COMPUTATION OF $\mathbf{C}_{t_0,t_f}$.

Data: $\mathbf{K}$,$\mathbf{T}$, t_0, t_f, $\mathbf{U}$
1 $S^r(t_f) = \{s_i \in S(t_f) | s_i \subseteq \mathbf{T}\}$;
2 $S^u(t_f) = \{s_i \in S(t_f) | s_i \cap \mathbf{T} = \emptyset\}$;
3 $S^i(t_f) = S(t_f) \setminus (S^r(t_f) \cup S^u(t_f))$;
4 **for** $t_k \leftarrow t_{f-1}$ **to** t_0 **do**
5 $\mathcal{L} = \emptyset$;
6 **forall the** $[\mathbf{u}_k] \in \mathbf{U}$ **do**
7 $\mathcal{L}.add(\mathbf{K})$;
8 **while** $\mathcal{L}$ *is not empty* **do**
9 $[\mathbf{x}] = \mathcal{L}.pop\_out()$;
10 $[\mathbf{x}]^* = \varphi(t_k, t_{k+1}; [\mathbf{x}], [\mathbf{u}_k])$;
11 **if** $[\mathbf{x}]^* \cap \mathbf{K} = \emptyset$ **then**
12 $\forall s_i \subseteq [\mathbf{x}], ADD(S^u(t_k), s_i)$;
13 **else if** $[\mathbf{x}]^* \subseteq S^r(t_{k+1})$ **then**
14 **if** $\phi([t_k, t_{k+1}]; [\mathbf{x}], [\mathbf{u}_k]) \subseteq \mathbf{K}$ **then**
15 $\forall s_i \subseteq [\mathbf{x}], ADD(S^r(t_k), s_i)$;
16 **else**
17 $\forall s_i \subseteq [\mathbf{x}], ADD(S^i(t_k), s_i)$;
18 **else if** $[\mathbf{x}]^* \subseteq S^u(t_{k+1})$ **then**
19 $\forall s_i \subseteq [\mathbf{x}], ADD(S^u(t_k), s_i))$;
20 **else if** $[\mathbf{x}]$ *can be bisected* **then**
21 $([\mathbf{x}_1], [\mathbf{x}_2]) = BISECT([\mathbf{x}])$;
22 $\mathcal{L}.add([\mathbf{x}_1]), \mathcal{L}.add([\mathbf{x}_2])$;
23 **else**
24 $\forall s_i \subseteq [\mathbf{x}], ADD(S^i(t_k), s_i))$;

Result: $\{S(t_k)\}, t_k = t_0, \cdots, t_f$.

unreachable cells are the ones that do not intersect the target and the *indeterminate* cells are all the others ones (the cells intersecting the target without been included). Then lines 4 to 24, the others slice subsets are built. Line 6 it can be notices that all the possible control vectors are considered to determine the reachability of the cells. Line 8 to 24 a *Set Inversion Via Interval Analysis* approach (Jaulin and Walter, 1993) is used to determinate the reachability of the current slice cells. It can be noticed that the computation of φ line 10 is done using guaranteed numerical integration (VNODE-LP). Line 14, a cell can be reachable only if the trajectory is included in the state space $\mathbf{K}$. Usually the computation of the inclusion flow is based on the Banach fixed-point theorem and the application of the Picard-Lindelof operator (see (Berz and Makino, 1998; Nedialkov et al., 1999) for details). Line 21, the box is bisected among the grid. It means that, line 20, the box $[\mathbf{x}]$ can not be bisected if it contains only one cell. In other words, the algorithm stops when all the indeterminate boxes have a grid cell size.

The result of the algorithm has to be interpreted as

$$\begin{aligned}\mathbf{C}^{+}_{t_0,t_f} &= S^r(t_0) \cup S^i(t_0)\\ \mathbf{C}^{-}_{t_0,t_f} &= S^r(t_0)\end{aligned} \tag{14}$$

The Figure 2 represents three cases of the Algorithm 1:

- $[\mathbf{x}_1]^* \subset S^u(t_{k+1})$, then $\{s_i \subseteq S(t_k) | s_i \subseteq [\mathbf{x}_1]\}$ can be added to $S^u(t_k)$ (Line 19 of the algorithm),
- $[\mathbf{x}_2]^* \subset S^r(t_{k+1})$, then $\{s_i \subseteq S(t_k) | s_i \subseteq [\mathbf{x}_2]\}$ can be added to $S^r(t_k)$ $[\mathbf{x}_2]^*$ (Line 15 of the algorithm),
- $[\mathbf{x}_3]^*$ is neither included in $S^u(t_{k+1})$ or $S^r(t_{k+1})$, and the box $[\mathbf{x}_3]$ is too small to be bisected, thus $\{s_i \in S(t_k) | s_i \subseteq [\mathbf{x}_3]\}$ can be added to $S^i(t_k)$ (Line 24 of the algorithm).

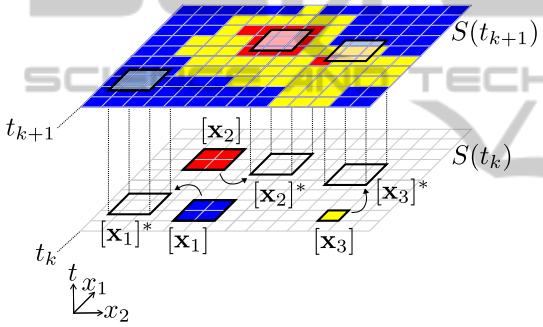

Figure 2: The reachability of the cells $s_i \subseteq S(t_{k+1})$ are used to build the subsets of the slice $S(t_k)$. Denote that $[\mathbf{x}_i]^* = \varphi(t_k, t_{k+1}; [\mathbf{x}_i], [\mathbf{u}_k]), i = 1,2,3$, with $[\mathbf{x}_1]^* \subset S^u(t_{k+1})$, $[\mathbf{x}_2]^* \subset S^r(t_{k+1})$ and $[\mathbf{x}_3]^*$ is neither included in $S^u(t_{k+1})$ or $S^r(t_{k+1})$.

3.2 The ADD algorithm

The slide's cell reachability is updated regards to all the possible control vectors $[\mathbf{u}_k] \in \mathbf{U}$. For a given cell, the reachability information can be different considering two different control vectors. That is why it is needed to consider *priorities* for the update of the reachability information of a cell and thus the computing of the three subsets $S^r(t_k)$,$S^u(t_k)$ and $S^i(t_k)$. That is the purpose of the ADD function. For example if a control vector $[\mathbf{u}_{1,k}]$ leads to a *reachability* information for a cell $s_i \subseteq S(t_k)$ whereas a control vector $[\mathbf{u}_{2,k}]$ leads to a *indeterminate* information for the same cell, this cell s_i belongs to $S^r(t_k)$ (is *reachable*) because it has been proved that it exists a control vector, $[\mathbf{u}_{2,k}]$, that leads the cell to $S^r(t_{k+1})$. Figure 3 presents the several reachability information priorities.

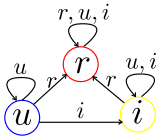

Figure 3: The reachability information priorities. A cell noted *unreachable* can be updated to *reachable* or *indeterminate*, a cell noted *indeterminate* can only be updated to *reachable* and a cell noted *reachable* can not be updated at all.

4 EXPERIMENTATION

In order to validate the proposed method the algorithm has been implemented in C++ using VNODE-LP library. Be considered the following two-dimensional system

$$\begin{cases}\dot{x}_1 &= x_2 + v \times \cos(\theta),\\ \dot{x}_2 &= \sin(x_1) + v \times \sin(\theta),\end{cases} \tag{15}$$

the following input vector

$$\begin{cases}\mathbf{U} &= [\mathbf{u}_{1,k}] \cup [\mathbf{u}_{2,k}] \cup [\mathbf{u}_{3,k}]\\ [\mathbf{u}_{i,k}] &= ([v_i], \theta_i),\\ [\mathbf{u}_{1,k}] &= ([-0.25, 0.25], \pi/2),\\ [\mathbf{u}_{2,k}] &= ([3.75, 4.25], \pi/2),\\ [\mathbf{u}_{3,k}] &= ([9.75, 10.25], \pi/2),\end{cases} \tag{16}$$

the following parameters

$$\begin{cases}\delta_t &= 0.5,\\ t_0 &= 0,\ t_f = 10,\\ \delta_{x_1} &= 0.5,\ \delta_{x_2} = 0.5,\\ \mathbf{K} &= ([-30, 30], [-30, 30]),\end{cases} \tag{17}$$

and the following target

$$\mathbf{T} = ([-15.1, -9.9], [1.9, 7.1]). \tag{18}$$

The Figure 4 shows four slices. The PC we use has two processors (Intel(R) Core(TM)2 CPU 6420 @ 2.13 Ghz), and it takes 1457s (24min 17s) to compute all the slices $S(t_k)$. The details of the computation time are presented in the Table 1.
The slice $S(0)$ provides the $\mathbf{C}^{+}_{t_0,t_f} = S^r(0) \cup S^i(0)$ and $\mathbf{C}^{-}_{t_0,t_f} = S^r(0)$ characterizations of $\mathbf{C}_{t_0,t_f}$. Note that it is possible to increase the precision of the approximation of $\mathbf{C}_{t_0,t_f}$ by reducing the values of δ_{x_1} and δ_{x_2}.

5 DISCRETE OPTIMAL CONTROL ENCLOSURE

The previous algorithm computes two guaranteed approximations $\mathbf{C}^{-}_{t_0,t_f}$ and $\mathbf{C}^{+}_{t_0,t_f}$ of $\mathbf{C}_{t_0,t_f}$. It is possi-

Table 1: Slices computation time.

slice	$S(9.5)$	$S(9)$	$S(8.5)$	$S(8)$
time (s)	2.56	5.55	11.95	22.6
slice	$S(7.5)$	$S(7)$	$S(6.5)$	$S(6)$
time (s)	40.66	61.78	71.86	75.6
slice	$S(5.5)$	$S(5)$	$S(4.5)$	$S(4)$
time (s)	76.41	79.6	82.88	87.89
slice	$S(3.5)$	$S(3)$	$S(2.5)$	$S(2)$
time (s)	93.96	98.59	102.79	105.87
slice	$S(1.5)$	$S(1)$	$S(0.5)$	$S(0)$
time (s)	106.78	108.5	109.23	111.73

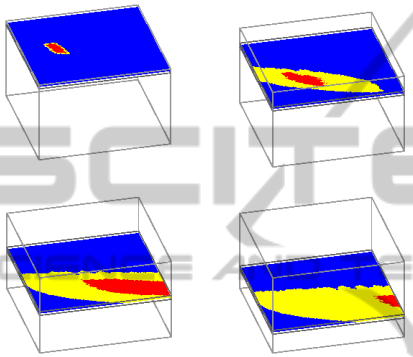

Figure 4: Four slices: (from left top to right bottom) $S(9.5)$, $S(7.5)$, $S(5)$ and $S(2.5)$.

ble to extend this algorithm to be able to deal with optimal control. Considering a given cost function $J(\mathbf{x}(t),\mathbf{u}(t))$, the idea of the algorithm is to enclose the cost of the trajectories that allow to reach a cell $s_i \subseteq S(t_{k+1})$ from a cell $s_j \subseteq S(t_k)$. Note that the input is still assumed to be continuous and bounded over $[t_{k-1},t_k]$.

To simplify we assume that the cost function to minimize is

$$J = \int \mathbf{u}(t)^2 dt. \tag{19}$$

It can obviously be extended to other cost functions. Note that is J is dependant of the state $\mathbf{x}$, the cost of an input vector between two time steps can still be computed using interval arithmetic and the evaluation of the trajectory $\phi([t_k,t_{k+1}],\mathbf{x}_k,\mathbf{u}_k)$.

Instead of characterizing the cells with the reachability of the target this section provides a method to add the input control that could be used to reach the target. The modifications of the previous algorithm are presented in Subsection 5.1, Subsection 5.2 details how to build a graph using the added control vectors, and Subsection 5.3 explains how to use the graph to enclose the optimal control input. Note that the found enclosure corresponds to the enclosure of the optimal trajectory assuming that the input vector is bounded between each time steps.

5.1 The Algorithm Modifications

The idea is to use the $\mathbf{C}_{t_0,t_f}$ computation to enclose the optimal trajectory from an initial state $[\mathbf{x}_0] \in \mathbf{C}_{t_0,t_f}$ to the target $\mathbf{T}$. To this end it is needed to slightly modify the presented algorithm. The purpose of this new algorithm is to define for all the cells $s_i \subseteq S(t_k)$ a set of input vectors $\mathbf{U}(s_i)$ that leads the cell to $S^r(t_{k+1})$ or $S^i(t_{k+1})$ (Figure 5):

$$\mathbf{U}(s_i) = \{[\mathbf{u}_k] \in \mathbf{U} | \varphi(t_k,t_{k+1};s_i,[\mathbf{u}_k]) \not\subseteq S^u(t_{k+1})\}, \tag{20}$$

with $s_i \in S(t_k)$, $t_k < t_f$.

Each time a cell $s_i \subseteq S(t_k)$ may be added to $S^i(t_k)$ or $S^r(t_k)$ (lines 15,17 and 24 of the Algorithm 1) the current input vector $[\mathbf{u}_k]$ has to be added to $\mathbf{U}(s_i)$.

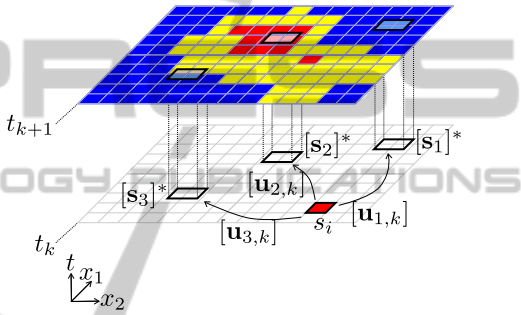

Figure 5: Example of added control vectors for a cell $s_i \subseteq S(t_k)$: $\mathbf{U}(s_i) = \{[\mathbf{u}_{2,k}],[\mathbf{u}_{3,k}]\}$, $[\mathbf{u}_{1,k}]$ is not relevant since it leads to $S^u(t_{k+1})$. Note that $[\mathbf{s}_j]^* = \varphi(t_k,t_{k+1};s_i,[\mathbf{u}_{j,k}]), j = 1,2,3$.

Considering the cost function it is possible to associate a cost $J([\mathbf{u}_k])$ to each control vector $[\mathbf{u}_k] \in \mathbf{U}(s_i)$, $s_i \subseteq S(t_k)$. In the following a control vector $[\mathbf{u}_k]$ will be abusively associated to its cost $J([\mathbf{u}_k])$.

5.2 A Graph Building

Given an initial state $[\mathbf{x}_0] \in \mathbf{C}_{t_0,t_f}$, the idea is to build a graph starting with a node $n_0(t_0)$ and ending with a node $n_\mathbf{T}(t_f)$ (Figure 7), such as

$$\begin{aligned} n_0(t_0) &= \{s_i \subseteq S(t_0) | s_i \cap [\mathbf{x}_0] \neq \emptyset\} \\ n_\mathbf{T}(t_f) &= \{s_i \subseteq S(t_f) | s_i \cap \mathbf{T} \neq \emptyset\} \end{aligned} \tag{21}$$

N defines the set of nodes of the graph. A node $n_i(t_k) \in N$ is defined by a set of cells $s_i \subseteq S(t_k)$. Two nodes $n_i(t_k)$ and $n_j(t_{k+1})$ are linked if

$$\exists [\mathbf{u}(t_k)] \in \mathbf{U} | \forall s_i \in n_i(t_k), \varphi(t_k,t_{k+1};s_i,[\mathbf{u}_k]) \subseteq n_j(t_{k+1}) \tag{22}$$

with $J([\mathbf{u}_k])$ the weight of the edge that links the two nodes $n_i(t_k)$ and $n_j(t_{k+1})$.

It is possible to define a set of control vector $\mathbf{U}(n_i(t_k))$ for a node $n_i(t_k) \in N$ corresponding to all the control vectors $\mathbf{U}(s_i)$ of all the cells $s_i \subseteq n_i(t_k)$:

$$\mathbf{U}(n_i(t_k)) = \{\mathbf{U}(s_i), \forall s_i \in n_i(t_k)\}. \tag{23}$$

Note that for efficiency reason it is recommended to avoid control vector redundancy in $\mathbf{U}(n_i(t_k))$, $\forall n_i(t_k) \in N$ (otherwise identical nodes will appear several times in the graph).
The graph is built from $n_0(t_0)$ to $n_\mathbf{T}(t_f)$ using the corresponding edge sets. Algorithm 2 details the nodes building and the Figure 6 presents an example of graph building.

Algorithm 2: NODES.

Data: S, $n_0(t_0)$, $n_\mathbf{T}(t_f)$
$\mathcal{L} = n_0(t_0), N = \emptyset$;
while $\mathcal{L}$ *is not empty* **do**
  $n_i(t_k) = \mathcal{L}.pop\_out()$;
  $N.add(n_i(t_k))$;
  if $t_k < t_f$ **then**
    for *all* $[\mathbf{u}_k] \in \mathbf{U}(n_i(t_k))$ **do**
      $n_i(t_{k+1}) = \{s_i \subseteq S(t_{k+1}) |$ $\varphi(t_k, t_{k+1}; n_i(t_k), [\mathbf{u}_k]) \cap s_i \neq \emptyset\}$;
      $\mathcal{L}.add(n_i(t_{k+1}))$;
$N.add(n_\mathbf{T}(t_f))$;
Result: N.

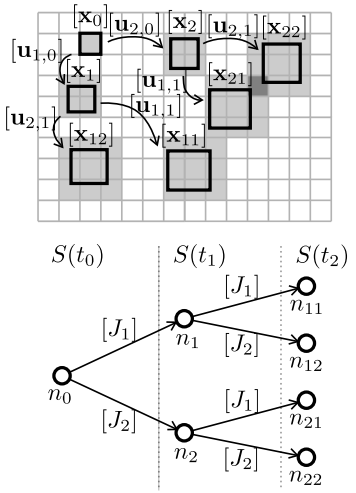

Figure 6: Example of graph building. Starting from a node $n_0(t_0) = \{s_i \subseteq S(t_0) | s_i \cap [\mathbf{x}_0] \neq \emptyset\}$, two nodes are computed: $n_1(t_1) = \{s_i \subseteq S(t_1) | s_i \cap [\mathbf{x}_1] \neq \emptyset\}$ and $n_2(t_1) = \{s_i \subseteq S(t_1) | s_i \cap [\mathbf{x}_2] \neq \emptyset\}$, with $[\mathbf{x}_1] = \varphi(t_0, t_1; n_0(t_0), [\mathbf{u}_{1,0}])$ and $[\mathbf{x}_2] = \varphi(t_0, t_1; n_0(t_0), [\mathbf{u}_{2,0}])$. The same principle is repeated for the other nodes. It can be noticed that
- the top figure corresponds to the superimposition of the three slices $S(t_0)$, $S(t_1)$, and $S(t_2)$,
- some cells can belong to different nodes, as the dark grey cell is attached to the node $n_{22}(t_2)$ and $n_{21}(t_2)$.

The computed graph corresponds to all the possible trajectories of the system that may lead to the target at t_f from an initial state $[\mathbf{x}_0]$ at t_0. A priori it encloses the optimal trajectory. This graph has a particularity: as the nodes of the graph are cell sets, it can be associated a reachability information for each node $n_i(t_k) \in N$

- a node $n_i(t_k)$ is *reachable* if all the cells $s_i \in n_i(t_k)$ are *reachable*,
- a node $n_i(t_k)$ is *indeterminate* if at least one cell $s_i \in n_i(t_k)$ is not *reachable*.

This can be extended to the paths of the graph

- a path is *reachable* if all the nodes of the path are *reachable*,
- a path is *indeterminate* if at least one node of the path is *indeterminate*. It can be noticed that an *indeterminate* path may correspond to a trajectory that does not exist considering the system. They have to be considered carefully.

5.3 Exploitation of the Graph

Using this graph, it is possible, with a shortest path algorithm, to compute two informations:

- an enclosure of the optimal control vector to reach the target $\mathbf{T}$ from an initial state $[\mathbf{x}_0] \in \mathbf{C}_{t_0,t_f}$,
- an evaluation of the cost of this control vector.

The chosen shortest path algorithm is a generalization of the Dijkstra algorithm (Dijkstra, 1971). The classical Dijkstra algorithm is presented Algorithm 3. The input of this algorithm is a graph G, composed by a set of nodes N linked to each others with weighted edges. $J(n_i)$ corresponds to the weight of the node n_i and $J(n_i, n_j)$ corresponds to the weight of the edge linking the nodes n_i and n_j (in our case it corresponds to the cost of the control vector from n_i to n_j). The Dijkstra algorithm can easily be extended for edges with interval weights as the $min()$ function can be extended to intervals ($min([\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}]) = [min(\underline{x_1}, \underline{x_2}), min(\overline{x_1}, \overline{x_2})]$ e.g. $min([3,9],[5,7]) = [3,7]$). Note that with interval weights it may not be possible to choose between two paths, in this case, both paths are solutions.

As some paths may not correspond to possible trajectories of the studied system (*indeterminate* paths), a *reachable* sub-graph has to be considered. Be a graph G, it is possible to build a sub-graph G_r defined by all the *reachable* paths of G. In this case G corresponds to all the trajectories that may lead the system to the target and G_r corresponds to all the guaranteed trajectories that lead the system the target from the initial state.

Processing an interval Dijkstra algorithm over G_r it is possible to find a set $\mathbf{P}_r^*$ of shortest guaranteed paths for this sub-graph.

$$\mathbf{P}_r^* = \{\text{shortest paths } P \in G_r\} \tag{24}$$

Algorithm 3: Dijkstra Algorithm.

Data: G
initialize the nodes as *unmarked*;
$\forall n_i \in N, J(n_i) = +\infty$;
$L(n_0) = 0$;
while *it exists an unmarked node* **do**
  n_L = the unmarked node with the lowest J;
  note n_L as *marked*;
  for *all unmarked nodes* n_U *linked to* n_L **do**
    $J(n_U) = min(J(n_U), J(n_L) + J(n_L, n_U))$;
Result: weighted node set N.

As mentioned before, when it is not possible to determinate if a path is shorter than an other one the two paths have to be kept as solution. For example if a path P_1 has a cost $J(P_1) = [20, 25]$ and a path P_2 has a cost $J(P_2) = [22, 30]$, it is not possible to determinate which path is shorter ($J(P_1) \not< J(P_2)$ since $25 > 22$, and $J(P_2) \not< J(P_1)$ since $22 > 20$). In this case the two paths P_1 and P_2 have to be kept as they may both be the shortest path.
The paths so obtained are guaranteed to exist, but may not be the shortest paths considering the graph G. The idea is to consider $\mathbf{P}_r^*$ as an upper bound of the shortest path of G.
Knowing $\mathbf{P}_r^*$ and processing an other interval Dijkstra algorithm over G it is possible to finally find the path set $\mathbf{P}^*$ that encloses the shortest path of G:

$$\mathbf{P}^* = \{\text{paths } P \in G \text{ that may be better than } \mathbf{P}_r^*\} \cup \mathbf{P}_r^* \quad (25)$$

Note that a path may be better than an other, if it is not possible to determinate which path is shorter.

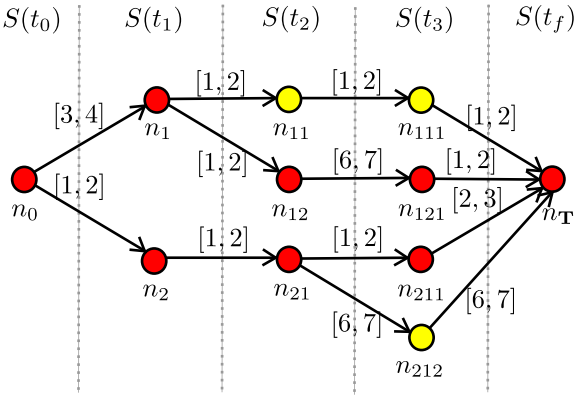

Figure 7: A graph example. Each node corresponds to a set of cells. The edges have interval weights corresponding to the costs $J([\mathbf{u}_k])$ of the control vectors $[\mathbf{u}_k] \in \mathbf{U}$. Note that $n_0, n_1, n_2, n_{12}, n_{21}, n_{121}, n_{211}$ are *reachable* nodes whereas n_{11}, n_{111}, n_{212} are *indeterminate* nodes.

Example of the Figure 7
Considering the graph G of the Figure 7, the following *reachable* sub-graph can be computed:

- $N_r = \{n_0, n_1, n_2, n_{12}, n_{21}, n_{121}, n_{211}, n_{\mathbf{T}}\}$ corresponding to the *reachable* nodes and edges,

with N_r the node set of G_r.
For the reader information here are the computation of all the path costs:

- $J(P(n_0, n_1, n_{11}, n_{111}, n_{\mathbf{T}})) = [6, 10]$
- $J(P(n_0, n_1, n_{12}, n_{121}, n_{\mathbf{T}})) = [11, 15]$
- $J(P(n_0, n_2, n_{21}, n_{211}, n_{\mathbf{T}})) = [5, 9]$
- $J(P(n_0, n_2, n_{21}, n_{212}, n_{\mathbf{T}})) = [14, 18]$

Using the interval Dijkstra algorithm it is possible to evaluate the optimal paths of the graph G_r, $\mathbf{P}_r^* = \{P(n_0, n_2, n_{21}, n_{211}, n_f)\}$. By processing an other interval Dijkstra algorithm over G and keeping only the paths that may be better that $\mathbf{P}_r^*$ we obtain $\mathbf{P}^* = \{P(n_0, n_1, n_{11}, n_{111}, n_{\mathbf{T}})\} \cup \mathbf{P}_r^*$. It can be concluded that the optimal path of the system for this example is included in $\mathbf{P}^* = \{P(n_0, n_1, n_{11}, n_{111}, n_f), P(n_0, n_2, n_{21}, n_{211}, n_{\mathbf{T}})\}$ and has a cost $J(\mathbf{P}^*) = [5, 9]$.

6 CONCLUSIONS

In this paper, we have introduced two interval-based algorithms. The first one allows, given a step time, to compute an evaluation of the subset $\mathbf{C}_{t_0,t_f}$ of initial states from which there exists at least one trajectory of the system reaching the target $\mathbf{T}$ in finite time t_f from a time t_0, assuming that the input vector is continuous and bounded over a time $[t_{k-1}, t_k]$. The result of this work is an outer and inner characterisation of $\mathbf{C}_{t_0,t_f}$. Then adapting this work we have defined an other algorithm to deal with discrete optimal control characterization. This second algorithm computes a graph corresponding to all the possible discrete trajectories that might lead the system from an initial state to the target. Using a generalized Dijkstra algorithm, it is possible to use this graph in order to enclose the discrete optimal control vector and evaluate is cost. As future work we are planing to modify the algorithm in order to be independent of the time step.

REFERENCES

Aubin, J.-P. (1990). A survey of viability theory. *SIAM Journal on Control and Optimization*, 28(4):749–788.
Aubin, J.-P. (2006). *Viability Theory*. Systems and Control. Springer Verlag.
Berz, M. and Makino, K. (1998). Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4:361–369.

Delanoue, N., Jaulin, L., Hardouin, L., and Lhommeau, M. (2009). Guaranteed characterization of capture basins of nonlinear state-space systems. In Filipe, J., Cetto, J., and Ferrier, J.-L., editors, *Informatics in Control, Automation and Robotics*, volume 24 of *Lecture Notes in Electrical Engineering*, pages 265–272. Springer Berlin Heidelberg.

Dijkstra, E. (1971). *EWD316: A Short Introduction to the Art of Programming*. Holland.

Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001). *Applied Interval Analysis*. Springer.

Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*.

Kirk, D. (2004). *Optimal Control Theory: An Introduction*. Dover books on engineering. Dover Publications, Incorporated.

Lhommeau, M., Jaulin, L., and Harouin, L. (2011). Capture basin approximation using interval analysis. *International Journal of Adaptative Control and Signal Processing*.

Moore, R. E. (1966). *Interval analysis*. Prentice-Hall series in automatic computation. Prentice-Hall.

Nedialkov, N. S., Jackson, K. R., and Corliss, G. F. (1999). Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21 – 68.