Acta Cybernetica **25** (2021) 85–99.

Validated Trajectory Tracking using Flatness

Olivier Mullier[ab] and Julien Alexandre dit Sandretto[ac]

Abstract

The problem of a safe trajectory tracking is addressed in this paper. The method consists in using the results of a validated path planner: a set of safe trajectories. It produces the set of controls to apply to remain inside this set of planned trajectories while avoiding static obstacles. The computation is performed using the differential flatness property of many dynamical systems. The method is illustrated in the case of the Dubins car model.

Introduction

In the context of cyber-physical systems, the problem of validated trajectory tracking is addressed. It consists in driving a controlled differiential system from an initial state region to a given target region while avoiding collisions with static obstacles. The general method relies on two steps: *(i)*, a path planning provides one path from the initial state to the final state and, *(ii)*, a trajectory tracker computes the controls that will be given to the actual controlled system to follow this planned path. When the system is critical, it is mandatory to bring certainty on the non violation of the constraints on the system, even when uncertainties on the model and/or the control to be applied occur.

Related Work

A classical way for the path planning is to use the RRT algorithm [11] (Rapidly-exploring Random Tree) and its variants. In [21, 20, 2], it has been successfully adapted to the case of critical systems where guarantees on the result are mandatory and where uncertainties have to be considered. This adaptation of the RRT algorithm uses of a set-membership computation, interval analysis, to no longer produce a trajectory to track but a set of possible trajectories with the property that no static obstacle can collides with the system when in this set. The generic algorithm is then called the box[1] RRT algorithm [21]. The differential flatness

[a]ENSTA PARIS, 828 Boulevard des maréchaux, 91120 Palaiseau, France, E-mail: {lastname}@ensta.fr

[b]ORCID: https://orcid.org/0000-0003-1439-746X

[c]ORCID: https://orcid.org/0000-0002-6185-2480

[1]box is refering to the cartesian product of intervals, see Section 1.1.

DOI: 10.14232/actacyb.285729

of some controlled systems is well known and studied for the control of nonlinear systems (see, *e.g.* [13, 5, 6, 15]). Flatness has already been successfully applied to trajectory tracking in the case of discrete time systems [18].

Contribution

The work here considers that a previous computation of a set of validated path planner is performed and it results in a set of safe trajectories that avoid any static forbidden area. From this set, one particular path is chosen to be tracked, it is done with the computation of a cubic Hermite spline from the system position and the desired trajectory translated in the flat output space. The controls are then produced using an endogenous dynamic feedback using flatness.

This work is presented as follows. Section 1 recalls the notions necessary to the presentation of our work: interval analysis, validated numerical integration of controlled systems, box RRT algorithm and cubic Hermite spline computation. Section 2 contains the main results of our work that is the validated tracking of a trajectory guarranteed to avoid any static obstacle. It is then illustrated in Section 3 with the computation of the controls given to a robot to drive from a set of initial states to a target area using the Dubins car model. Conclusion and discussions ends our work presentation.

1 Preliminaries

This section is dedicated to the preliminary results our work is based on and introduces the notions that are used through this article. Some recalls on interval analysis and validated numerical integration scheme, differential flatness, trajectory planning and cubic Hermite spline are given.

1.1 Interval Analysis

Interval analysis [17] is a method designed to produce outer-approximation of the set of possible values for variables occurring in some computations in a sound manner. Hereafter, an interval is denoted $[x] = [\underline{x}, \overline{x}]$ with $\underline{x} \leqslant \overline{x}$ and the set of intervals is $\mathbb{IR} = \{[x] = [\underline{x}, \overline{x}] \mid \underline{x}, \overline{x} \in \mathbb{R},\ \underline{x} \leqslant \overline{x}\}$. The Cartesian product of intervals $[\mathbf{x}] \in \mathbb{IR}^n$ is a box (through this paper, vectors are represented in bold font). The main result of interval analysis is its fundamental theorem [16] stating that the evaluation of an expression using intervals leads to an outer-approximation of the resulting set of values for this expression whatever the values considered in the intervals.

For a given function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a box $[\mathbf{x}] \subset \mathbb{R}^n$, an interval inclusion function also known as interval extension $[f]$ of f can be defined and an evaluation of $[f]$ over $[\mathbf{x}]$ gives a box $[\mathbf{y}]$ such that $(\forall \mathbf{x} \in [\mathbf{x}])(\exists \mathbf{y} \in [\mathbf{y}])(\mathbf{y} = f(\mathbf{x}))$ or equivalently, the box $[\mathbf{y}]$ contains $\{f(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\}$ the range of f over $[x]$. Examples of interval extension are the natural interval extension where interval arithmetic is used to

replace all operations defining a function to its interval counterpart and the mean value extension where the function to be extended is first linearized on a point and a natural extension of the resulting function is applied. The interested reader can refer, for example, to [17, 10] and references therein for more details.

1.2 Validated Numerical Integration of Controlled Systems

In our work, we deal with controlled differential systems of the form

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \\ \mathbf{x}(0) \in [\mathbf{x_0}] \end{cases} \tag{1}$$

with $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ a smooth function, $\mathbf{x} \in \mathbb{R}^n$ the state vector of the system and $\mathbf{u} \in [\mathbf{u}] \subseteq \mathbb{R}^m$ the control vector. The problem to integrate the controlled system in Eq. (1) is to compute the value of the state vector at a time t, $\mathbf{x}(t, x_0, u)$ with $x_0 \in [x_0]$ and $\mathbf{u} \in [\mathbf{u}]$. By considering $\mathbf{u}$ as constant during the time the system is integrated, the problem in Equation (1) corresponds to solving the (ordinary) differential equation

$$\begin{cases} \dot{\mathbf{x}} = f_{\mathbf{u}}(\mathbf{x}) \\ \mathbf{x}(0) \in [\mathbf{x_0}] \end{cases} \tag{2}$$

with $f_{\mathbf{u}}$ being a function parameterized by the the control $\mathbf{u}$. The use of validated numerical integration on ODE for the problem in Equation (2) allows the design of an interval inclusion function of $\mathbf{x}(t, x_0, u)$. It provides the computation of $[\mathbf{x}]\left(t; [\mathbf{x_0}], [\mathbf{u}]\right)$ which stands for an outer approximation of the solution of the problem in Equation (1): $\{\mathbf{x}(t; \mathbf{x_0}, \mathbf{u}), \mathbf{x_0} \in [\mathbf{x_0}], \mathbf{u} \in [\mathbf{u}]\}$. It corresponds to the set of values that can take the state $\mathbf{x}$ at time t starting from any point $\mathbf{x}_0 \in [\mathbf{x_0}]$ for all controls $\mathbf{u} \in [\mathbf{u}]$ applied to the system. Any bounded uncertainty in the model can also be handled by representing it with a time constant parameter as done with the control $\mathbf{u}$ in Eq. (2).

To integrate the system until time t, a sequence of time instants $t_1, \ldots, t_n$ such that $t_1 < \cdots < t_n = t$ and a sequence of boxes $[\mathbf{x_1}], \ldots, [\mathbf{x_n}]$ such that $\mathbf{x}(t_{i+1}; [\mathbf{x_i}], [\mathbf{u}]) \subseteq [\mathbf{x_{i+1}}]$, $\forall i \in \{0, \ldots, n-1\}$ are computed with $[\mathbf{x}_i]$ an outer approximation of the set of the state vectors at time t_i. From the box $[\mathbf{x_i}]$, computing the box $[\mathbf{x_{i+1}}]$ is a classical 2-step method (see [14, 19]):

Phase 1 (Picard-Lindelof operator) compute an a priori enclosure $[\mathbf{x}_{i,i+1}]$ of the set

$$\{\mathbf{x}(t_k; \mathbf{x}_i, \mathbf{u}) \mid t_k \in [t_i, t_{i+1}], \mathbf{x}_i \in [\mathbf{x_i}], \mathbf{u} \in [\mathbf{u}]\} \subseteq [\mathbf{x}_{i,i+1}]$$

such that $\mathbf{x}(t_k; [\mathbf{x}_i], [\mathbf{u}])$ is guaranteed to exist and is unique,

Phase 2 compute an enclosure $[\mathbf{x}_{i+1}]$ of the solution at time t_{i+1}.

By repeating this process iteratively, we are able to produce an outer approximation of the system described in Equation (1) (see Figure 1). This scheme is used by the box RRT algorithm to compute a set of validated paths.

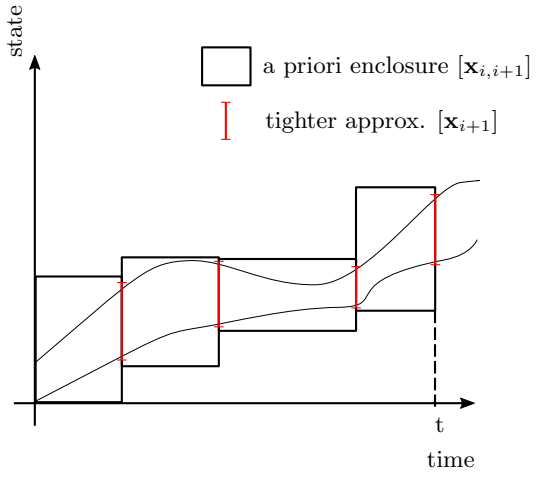

Figure 1: Illustration of the two-step method for the validated numerical integration of dynamical systems. $i = 1, ..., 5$.

1.3 The Box RRT Algorithm

The first part of the proposed method is to compute a set of validated trajectories. The box RRT algorithm and its improvements [2, 20] is a set of motion planner for robotic vehicles that guarantees to avoid static obstacles. It uses the data structures of the Rapidly-exploring Random Trees (RRT) to explore the state space. A tree of random subpaths is constructed until the goal is reached while any static obstacle is avoided. When the algorithm ends successfully, it provides, among other information, a set of boxes guaranteed to avoid any defined static obstacle. The result is in the form of a list of boxes $\{[x_1], \ldots, [x_N]\}$ all sorted by the time they are reached (see, for example, Figure 5 representing an example of the result of the algorithm described thereafter in Section 3). The box RRT algorithm provides also the control used during the seek of the set of trajectories but this information is not mandatory to the use of our method afterwards. Indeed, the box RRT can be used with a simplified model for the controlled dynamical system and then the controls $\mathbf{u}$ it uses are no longer relevant for the trajectory tracking part.

1.4 Differential Flatness

The differential flatness of dynamical systems is a structural property that a (possibly nonlinear) differential system can have. A system is differentially flat if there exists a set of independent variables (equal in number to the dimension of the control vector) referred to as flat outputs such that all states and controls of the system can be expressed in terms of those flat outputs and a finite number of their successive time derivatives. The mathematical definition of differential flatness is provided in Definition 1.

Definition 1 (Differential flatness [6])**.** *The controlled dynamical system*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{3}$$

with $\mathbf{x} \in \mathbb{R}^n$*,* $\mathbf{u} \in \mathbb{R}^m$ *is flat if there exist the maps* $h : \mathbb{R}^n \times (\mathbb{R}^m)^{r+1} \to \mathbb{R}^m$*,* $\varphi : (\mathbb{R}^m)^r \to \mathbb{R}^n$ *and* $\psi : (\mathbb{R}^m)^{r+1} \to \mathbb{R}^m$ *such that*

$$\begin{aligned} \mathbf{z} &= h\left(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ldots, \mathbf{u}^{(r)}\right) \\ \mathbf{x} &= \varphi\left(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(r-1)}\right) \\ \mathbf{u} &= \psi\left(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(r-1)}, \mathbf{z}^{(r)}\right) \end{aligned}$$

A differentially flat system can be caracterized when considering a subset of its state and control vectors and their associated derivatives. As recalled in the following, a trajectory can be tracked in a flat output space. An example of differentially flat system is given hereafter in Section 3.2.

1.5 Trajectory Tracking

The differential flatness makes possible to control a system by only using the flat output. The next definition recalls the endogenous dynamic feedback [7] in the case of a differentially flat system.

Definition 2 (Endogenous dynamic feedback using flatness)**.** *We consider the flat system described in Eq.* (3) *with a flat output* $\mathbf{z}$ *and a given trajectory* $\mathbf{z}_d : \mathbb{R} \to \mathbb{R}^m$*. Once the flat system is differentiated enough to get an equation of the type*

$$\mathbf{z}^{(p)} = \omega, \tag{4}$$

the system can be stabilized around a trajectory by stabilizing the flat output with ω *defined as a particular control:*

$$\omega = \mathbf{z}_d^{(p)} - k_0(\mathbf{z} - \mathbf{z}_d) - k_1(\dot{\mathbf{z}} - \dot{\mathbf{z}}_d) - \cdots - k_{p-1}(\mathbf{z}^{(p-1)} - \mathbf{z}_d^{(p-1)}) \tag{5}$$

with $\mathbf{z}^{(i)}$ *the* $i-th$ *time derivative of* $\mathbf{z}$ *and* $k_0, \ldots, k_{p-1}$ *such that the polynomial* $s^p = k_{p-1}s^{p-1} + \cdots + k_1 s + k_0$ *has only roots with negative real part. It results in the tracking error converging and stable.*

The goal is now to construct this function $\mathbf{z}_d(t)$ providing a trajectory in the flat output space. In our context, it is done using the cubic Hermite splines.

1.6 The Cubic Hermite Splines

In order to use endogenous dynamic feedback using flatness, we have to compute one particular trajectory $\mathbf{z}_d$ from the initial state to the target the robot will have to follow. An intermediary desired goal is required to be reached iteratively until we reach the goal. The cubic Hermite splines [8, 4] are one way to achieve this (see, *e.g.* [12] for a use in path planning and [9] for one in trajectory tracking).

Definition 3 (Cubic Hermite Spline)**.** *Let $\phi : \mathbb{R} \to \mathbb{R}^n$ be a diffentiable function, t_0, $t_1 \in \mathbb{R}$ and two positions $\mathbf{p_0} = \phi(t_0)$ and $\mathbf{p_1} = \phi(t_1)$ with the derivatives $\mathbf{m_0} = \dot{\phi}(t_0)$ and $\mathbf{m_1} = \dot{\phi}(t_1)$ respectively, the cubic Hermite spline is*

$$P(t) = p\left(\frac{t - t_0}{t_1 - t_0}\right) \tag{6}$$

with

$$p(t) = h_{00}(t)\mathbf{p_0} + h_{10}(t)\mathbf{m_0}(t_1 - t_0) + h_{01}(t)\mathbf{p_1} + h_{11}(t)\mathbf{m_1}(t_1 - t_0) \tag{7}$$

and

$$\begin{aligned} h_{00}(t) &= 2t^3 - 3t^2 + 1 \\ h_{10}(t) &= t^3 - 2t^2 + t \\ h_{01}(t) &= -2t^3 + 3t^2 \\ h_{11}(t) &= t^3 - t^2. \end{aligned}$$

The error between the spline $P(t)$ and the true trajectory $\phi(t)$ can be easily computed as follows: there exists τ such that

$$\phi(t) - P(t) = \frac{\phi^{(k)}(\tau)}{k!} \prod_i (t - t_i)^{k_i} \tag{8}$$

with k the number of points, k_i the number of known derivatives + 1. In our case, the number of points is 2 and the number of derivatives is 2.

This error can be approximated using interval analysis as follow: in the time interval $[t_0, t_1]$ where the spline is defined, The error can be outer approximated by the interval function $[E]\,(t, [t_0, t_1])$ such that:

$$\phi(t) - P(t) \in [E]\,(t, [t_0, t_1]) = \frac{\left[\ddot{\phi}\right]([t_0, t_1])}{2}(t - t_0)^3(t - t_1)^3, \ \forall t \in [t_0, t_1] \tag{9}$$

with $\left[\ddot{\phi}\right]$ an interval inclusion function of the second time derivative of ϕ. In practice, the model being provided, the second (as well as the first) derivative over time of ϕ is also available by symbolically derivating the system over time prior to the execution of the method. A guaranteed cubic Hermite spline $[P]\,(t)$ then consists in adding this box error:

$$[P]\,(t) = P(t) + [E]\,(t, [t_0, t_1])$$

2 Validated Trajectory Tracking using Flatness

This section is dedicated to the main results on validated trajectory tracking using flatness. We consider a controlled dynamical system of the form described in Equation (1).

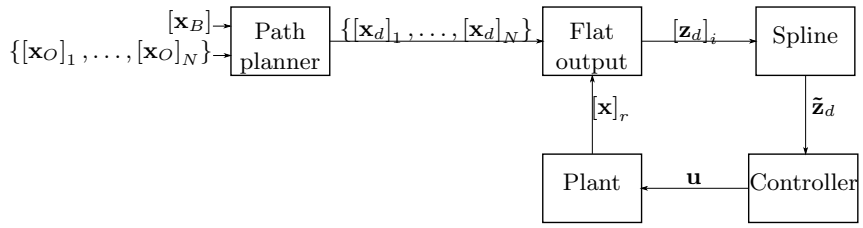

Figure 2: Block diagram representing the different components in our controller.

In this problem, the initial condition is assumed to be inside a set of values here defined as a box $[\mathbf{x}_0]$ and a goal $[\mathbf{x}_G]$ is also provided such that the goal is eventually reached at a bounded time $t_f \leqslant T$, as stated by the quantified proposition

$$(\exists u(t) : \mathbb{R} \to \mathbb{R}^m)(\exists t_f \in [0, T])(\mathbf{x}(t_f) \in [\mathbf{x}_G]). \tag{10}$$

A set of obstacles $[\mathbf{x}]_{O,1}, \ldots, [\mathbf{x}]_{O,N}$ is also defined and the trajectory $\mathbf{x}(t)$ verifying Eq. (10) must avoid them:

$$(\forall i = 1, \ldots, N)(\forall t \in [0, t_f])(\mathbf{x}(t) \notin [\mathbf{x}]_{O,i}). \tag{11}$$

The goal of trajectory tracking is then to produce the control function $u(t)$ such that the system follows a trajectory that respects the propositions in Equations (10) and (11) dealing with uncertainties maybe occurring in the model and the set of initial and final positions. A validated numerical integration scheme as described in Section 1.2 using interval analysis is used. We present here the method to produce the controls that allows a cyberphysical system to follow a path. The block diagram corresponding to the control strategy is given in Figure 2 and Algorithm 1 provides a sketch of the proposed method after the path planner provided its result.

Path planner The initial position of the robot $[\mathbf{x}_B]$, the goal $[\mathbf{x}_G]$ and the set of obstacles $\{[\mathbf{x}_O]_1, \ldots, [\mathbf{x}_O]_N\}$ are first given to the path planner which in return gives a set of guaranteed trajectories $\{[\mathbf{x}_d]_1, \ldots, [\mathbf{x}_d]_N\}$.

Flat output Using flatness, a box $[\mathbf{z}_d]_i$ in the flat output space is provided from the result of the path planner. This box $[\mathbf{z}_d]_i$ contains the current flat output of the robot $[\mathbf{z}_r]$ according to its current position $[\mathbf{x}_r]$.

Spline A cubic Hermite spline is then computed to return the next desired flat output $\tilde{\mathbf{z}}_d$ for the robot. It is illustrated in Fig. 3. A particular point $\tilde{\mathbf{z}}_{d;i+1} \in [\mathbf{z}_d]_{i+1}$ is chosen to be the desired flat output for the robot (we chose the midpoint of $[\mathbf{z}_d]_{i+1}$). The spline $P(t)$ is constructed from a point $\tilde{\mathbf{z}}_r$ in the flat output of the robot (we chose the midpoint of $[\mathbf{z}_r]$ as well) to the chosen point $\tilde{\mathbf{z}}_{d,i+1}$. The guarranteed spline $[P](t)$ is also computed and a verification on it is done to check is it remains in the union $[\mathbf{z}_d]_i \cup [\mathbf{z}_d]_{i+1}$. If the verification fails, a new endpoint for the spline is chosen, one inside the intersection $[\mathbf{z}_d]_i \cap [\mathbf{z}_d]_{i+1}$. An intermediary point $\tilde{\mathbf{z}}_d$ on the spline is chosen to be the next position to reach for the robot.

Input: $\{[\mathbf{z}_d]_i\}$: the result of the guaranteed path planner in the flat output space

while *the robot has not arrived* **do**

`// Flat output`

$[\mathbf{z}_r]$: the flat output of the robot from its current position $[\mathbf{x}_r]$

$[\mathbf{z}_d]_i$: box the robot is in $(([\mathbf{z}_r]) \subseteq [\mathbf{z}_d]_i)$

$\tilde{\mathbf{z}}_{d;i+1} = \text{mid}([\mathbf{z}_d]_{i+1})$

`// Spline`

$P(t)$: the Hermite spline between $\text{mid}([\mathbf{z}_r])$ and $\tilde{\mathbf{z}}_{d;i+1}$

$\tilde{\mathbf{z}}_d$: next reference point on the spline for the robot

`// Controller`

computation of the controls using dynamic feedback with flat output $\mathbf{z}_r$ and $\tilde{\mathbf{z}}_d$ for the robot to reach $\tilde{\mathbf{z}}_d$

`// Plant`

$[\mathbf{x}_r]$: new position of the robot after applying the controls

end

Algorithm 1: Sketch of the proposed method for the tracking of the trajectory provided by guaranteed path planner algorithm (sketch with correspondance with the block diagram in Fig. 2).

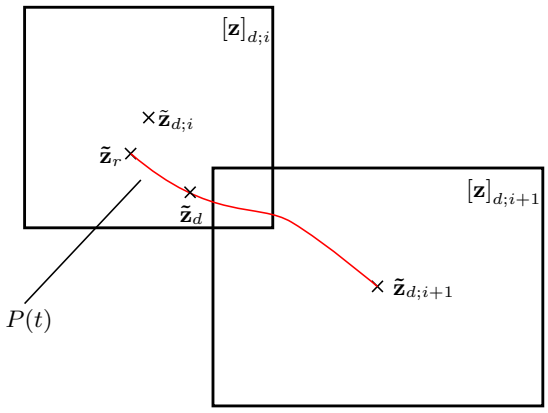

Figure 3: Computation of the next control from the computation of the cubic Hermite spline.

Controller All required information for the computation of a new control are provided: the spline provides the reference point $\tilde{\mathbf{z}}_d$ and the current flat output $[\mathbf{z}_r]$ of the robot is also known. Their time derivatives are directly known as well and a new control can be computed using the classical endogenous dynamic feedback using flatness as described in Definition 2. The control is then tested using a guaranteed numerical integration scheme to validate that the robot will not leave the set of guaranteed trajectories from the path planner in the same time horizon the controls will be applied to the robot. In the case the test fails, the time horizon and/or the choice on the reference point $\tilde{\mathbf{z}}_d$ should be modified. This change is not discussed in this article.

Plant The control is applied to the robot. It eventually provides the new position $[\mathbf{x}_r]$ of the robot and the process can start again until the goal $[\mathbf{x}_G]$ is reached ($[\mathbf{x}_r] \subseteq [\mathbf{x}_G]$).

The next section illustrates this method on a classical Dubins car model.

3 Experiments on the Dubins Car Model

In this section is given an example of the use of the method introduced in Section 2 on the Dubins car model.

3.1 Implementation

The experiment has been conducted using a C++ implementation of the trajectory tracking coupled with a C++ implementation of the box RRT algorithm. All the tests have been made by simulating the behaviour of the dynamical system in a guaranteed manner. The validated numerical integration part has been handled using the C++ library DynIbex[2] [1], a plugin of Ibex [3] on the Dubins car model, described in the following.

3.2 The Dubins Car Model

We consider the Dubins car model

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = u \end{cases} \tag{12}$$

with $(x, y)^T$ the position of the car and θ its angle to the coordinate system. The controls (u, v) are the angular and the longitudinal speeds of the vehicle respectively (see Figure 4 for an illustration of the Dubins car model).

It has been proved that this system is flat [13] and $z = (x, y)^T$ is a flat output. Indeed the state and the control variables can be written using x and y and their

[2] Available at: `https://perso.ensta-paris.fr/~chapoutot/dynibex/`

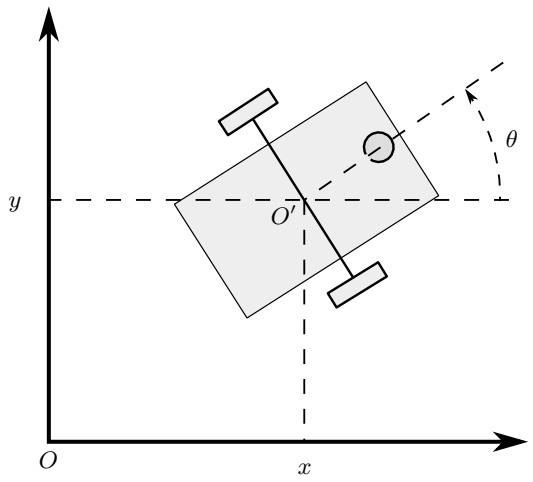

Figure 4: The Dubins car model.

derivatives:

$$\begin{cases} x = z_1 \\ y = z_2 \\ \theta = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right) = \tan^{-1}\left(\frac{\dot{z}_2}{\dot{z}_1}\right) \\ v = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{\dot{z}_1^2 + \dot{z}_2^2} \\ u = \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2} = \frac{\ddot{z}_2\dot{z}_1 - \ddot{z}_1\dot{z}_2}{\dot{z}_1^2 + \dot{z}_2^2}. \end{cases}$$

The set of available positions provided by the box RRT algorithm directly gives the flat output which is the position of the robot. In Figure 5 is given an example of the set of boxes returned by the box RRT algorithm. The black boxes represent the safe set of trajectories, the red ones are the obstacles, the blue one is the goal and the green dots are the centers of the black boxes (the safe set of trajectories). The set of trajectories given by the path planner (here the box RRT algoritm) is not optimal but it is not an issue for our method. We apply the controls produced by the method described in Section 2 and the results are shown in Figure 6. As in Figure 5 the result of the box RRT is still in black, the obstacle are in red and the goal is in blue. The green dots are the successive centers of the boxes containing the position of the robot after each application of the computed controls and the red dots are the reference points computed with the cubic Hermite splines. To make the figure more readable, only the centers of the position boxes of the robot are shown. At each step of the control to the goal, the box containing the robot $[\mathbf{x}_r]$ is checked to remain inside the black boxes of the box RRT algorithm as long as the guaranteed numerical simulation of the application of the control to the robot.

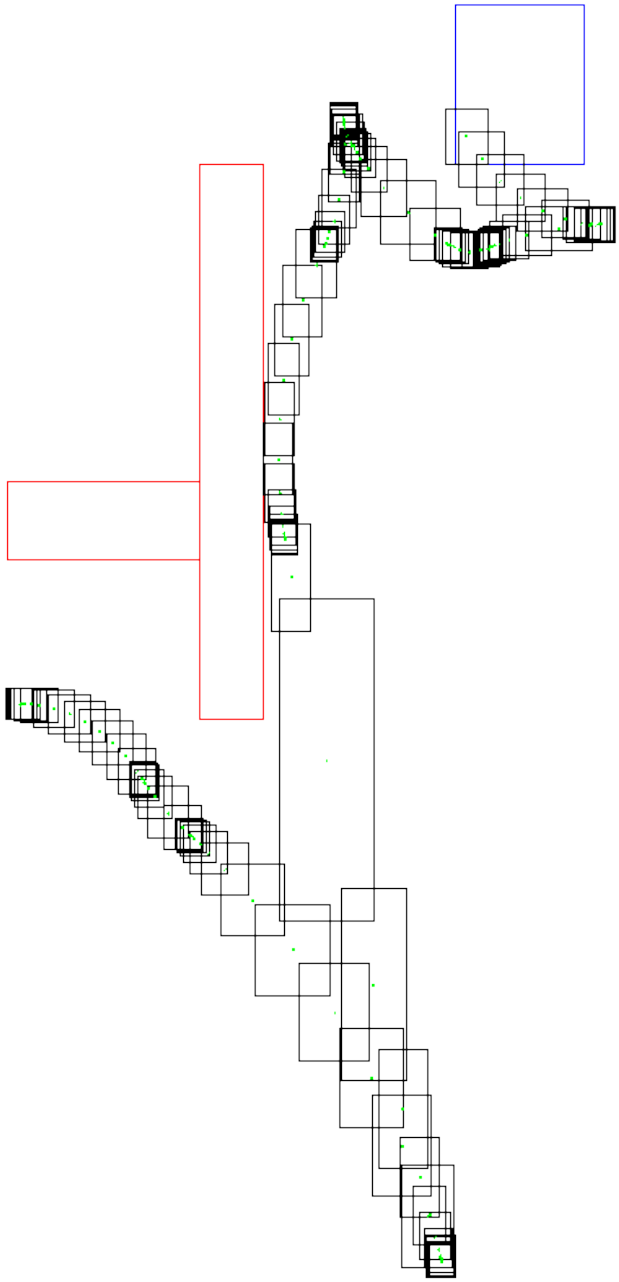

Figure 5: Example of the result provided by the box RRT algorithm. Red boxes: static obstacles ; blue: goal ; black: validated set of trajectories; green dots: center of the black boxes.

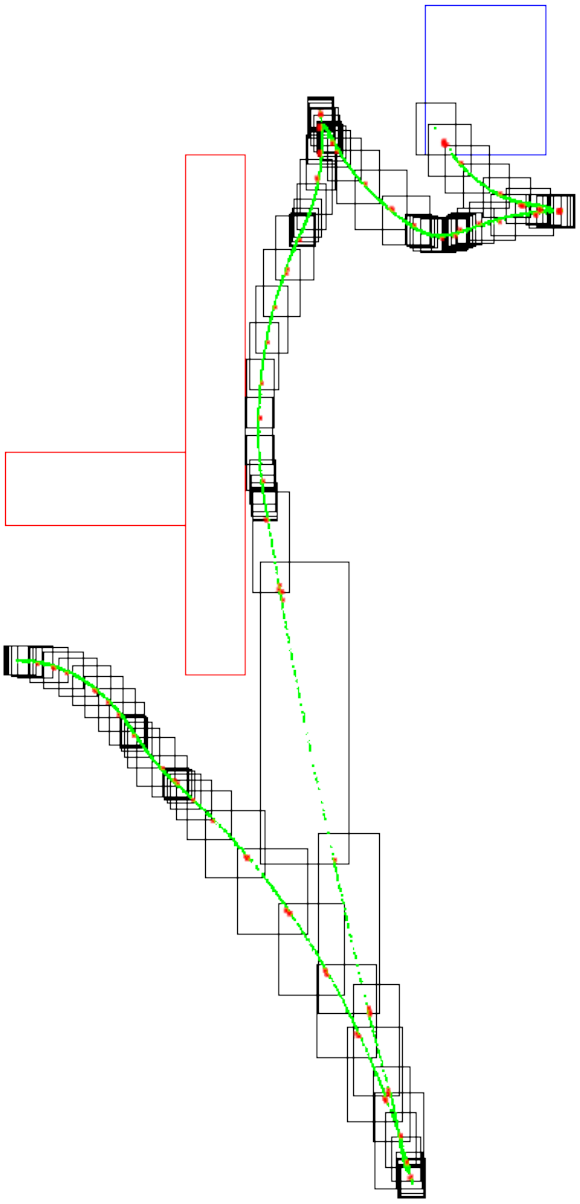

Figure 6: Example of the result of the controls computed to follow the set of trajectories from the box RRT algorithm (same as in Figure 5). Red boxes: static obstacles ; blue: goal ; black: validated set of trajectories; green dots: center of the robot position boxes. red dots (magnified): reference points from the cubic Hermite spline computation.

4 Conclusion

A validated trajectory tracking using flatness has been presented. From a guaranteed set of trajectory planner which provides an *a priori* set of guaranteed paths, our method uses flatness to perform the computation of the controls. The Intermediary reference points for the system are provided using guaranteed cubic Hermite splines. Eventually, the control to apply is proved to fulfill the requirements by simulating in a set-membership manner the use of the controls on the system. It allows to prove it remains inside the set of paths. This method has been illustrated with an example of a terrestrial robot with a flat system model. The next step will be to apply this method on a real robotic platform to validate the method in the context of an embedded system. An extra step is required to compute the control while the robot has not already reached its local goal provided by the cubic Hermite spline. Another improvement will be to take into account dynamical obstacles. It should be done by recalling the path planner each time a potential collision is detected. Eventually the method must be extended to the case of non flat systems.

Acknowledgment

This work has been partially supported by a DGA AID project.

References

[1] Alexandre dit Sandretto, Julien and Chapoutot, Alexandre. Validated explicit and implicit Runge–Kutta methods. *Reliable Computing*, 22(1):79–103, Jul 2016.

[2] Alexandre dit Sandretto, Julien, Chapoutot, Alexandre, and Mullier, Olivier. Formal verification of robotic behaviors in presence of bounded uncertainties. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 81–88. IEEE, 2017. DOI: `10.1109/IRC.2017.17`.

[3] Chabert, Gilles et al. Ibex, an interval-based explorer. `http://www.ibex-lib.org/`, 2007.

[4] De Boor, Carl. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978. DOI: `10.2307/2006241`.

[5] Fliess, M, Lévine, J, Martin, Ph, Ollivier, F, and Rouchon, P. Controlling nonlinear systems by flatness. In *Systems and Control in the Twenty-first Century*, pages 137–154. Springer, 1997. DOI: `10.1007/978-1-4612-4120-1_7`.

[6] Fliess, Michel, Lévine, Jean, Martin, Philippe, and Rouchon, Pierre. Flatness and defect of non-linear systems: introductory theory and examples.

International journal of control, 61(6):1327–1361, 1995. DOI: 10.1080/00207179508921959.

[7] Fliess, Michel, Lévine, Jean, Martin, Philippe, and Rouchon, Pierre. A Lie–Bäcklund approach to equivalence and flatness of nonlinear systems. *IEEE Transactions on Automatic Control*, 44(5), 1999. DOI: 10.1109/9.763209.

[8] Hermite, Charles. Sur la théorie des fonctions elliptiques. *Comptes rendus de l'Académie des sciences*, 57:613, 1863.

[9] Hintzen, Niels T, Piet, Gerjan J, and Brunel, Thomas. Improved estimation of trawling tracks using cubic Hermite spline interpolation of position registration data. *Fisheries Research*, 101(1-2):108–115, 2010. DOI: 10.1016/j.fishres.2009.09.014.

[10] Jaulin, Luc, Kieffer, Michel, Didrit, Olivier, and Walter, Eric. Interval analysis. In *Applied interval analysis*, pages 11–43. Springer, 2001. DOI: 10.1137/1009099.

[11] LaValle, Steven M. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[12] Lekkas, Anastasios M and Fossen, Thor I. Integral los path following for curved paths based on a monotone cubic Hermite spline parametrization. *IEEE Transactions on Control Systems Technology*, 22(6):2287–2301, 2014. DOI: 10.1109/TCST.2014.2306774.

[13] Levine, Jean. *Analysis and control of nonlinear systems: A flatness-based approach*. Springer Science & Business Media, 2009. DOI: 10.1007/978-3-642-00839-9.

[14] Lohner, Rudolf J. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In *Institute of mathematics and its applications conference series*, volume 39, pages 425–425. Oxford University Press, 1992.

[15] Ma, Dailiang, Xia, Yuanqing, Shen, Ganghui, Jia, Zhiqiang, and Li, Tianya. Flatness-based adaptive sliding mode tracking control for a quadrotor with disturbances. *Journal of the Franklin Institute*, 355(14):6300–6322, 2018. DOI: 10.1016/j.jfranklin.2018.06.018.

[16] Moore, R. E. *Interval Analysis*. Series in Automatic Computation. Prentice Hall, 1966. DOI: 10.1137/1009099.

[17] Moore, Ramon E, Kearfott, R Baker, and Cloud, Michael J. *Introduction to interval analysis*, volume 110. Siam, 2009. DOI: 10.1137/1.9780898717716.

[18] Mullier, Olivier and Courtial, Estelle. Set-membership computation of admissible controls for the trajectory tracking. *Reliable Computing*, 2017.

[19] Nedialkov, Nedialko S, Jackson, Kenneth R, and Corliss, George F. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999. DOI: 10.1016/S0096-3003(98)10083-8.

[20] Panchea, Adina, Chapoutot, Alexandre, and Filliat, David. Extended reliable robust motion planners. *56th IEEE Conference on Decision and Control*, Dec 2017. DOI: 10.1109/CDC.2017.8263805.

[21] Pepy, Romain, Kieffer, Michel, and Walter, Eric. Reliable robust path planner. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1655–1660. IEEE, 2008. DOI: 10.1109/IROS.2008.4650833.