Robust Optimal Control for Guidance of Autonomous Vehicles

Etienne Bertin

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 20XXIPPAXXXX

Thèse de doctorat

Robust Optimal Control for Guidance of Autonomous Vehicles

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École nationale supérieure de techniques avancées

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Signal, Images, Automatique et robotique

Thèse présentée et soutenue à Palaiseau, le 15 décembre 2022, par

ETIENNE BERTIN

Composition du Jury :

Nacim Ramdani Professeur, Université d'Orléans (PRISME)	Président et rapporteur
Jean-Baptiste Caillau Professeur, Université Côte d'Azur (CNRS, Inria, LJAD)	Rapporteur
Nicolas Delanoue Professeur associé, Université d'Angers	Examinateur
Carine Jauberthie Professeure associée, Université Toulouse III-Paul Sabatier	Examinatrice
Goran Frehse Professeur, ENSTA Paris (U2IS)	Directeur de thèse
Bruno Hérissé Ingénieur-Chercheur, ONERA (DTIS/NPGA)	Co-directeur de thèse
Julien Alexandre dit Sandretto Professeur Associé, ENSTA Paris (U2IS)	Invité
Alexandre Chapoutot Professeur Associé, ENSTA Paris (U2IS)	Invité

Contents

Remerciements

Ce manuscrit présente mes travaux de recherche entrepris en partenariat entre l'institut d'aérospatial ONERA et le labo U2IS de l'école d'ingénieur ENSTA Paris. Je commence donc par remercier ces deux organismes d'avoir mis du beurre dans les épinards.

Ma gratitude va ensuite à Nacim Ramdani et Jean-Baptiste Caillau, qui ont accepté d'évaluer mes travaux à mi-parcours, qui m'ont encouragé et qui ont accepté d'être rapporteurs de ce manuscrit. Je remercie aussi Nicolas Delanoue et Carine Jauberthie d'avoir accepté de faire partie du jury de soutenance de ma thèse.

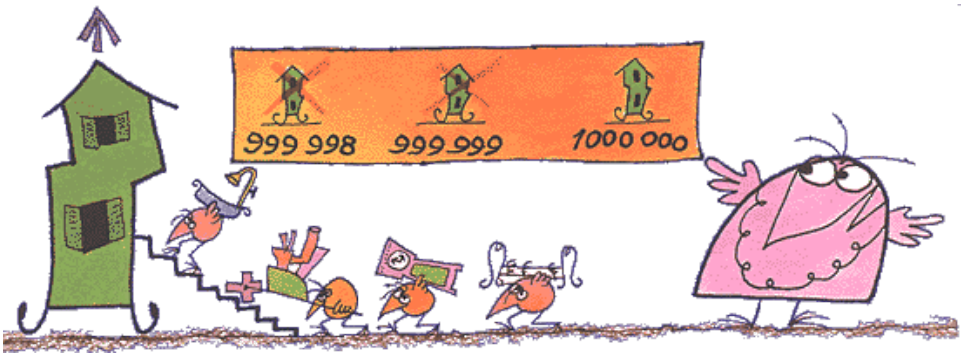
Atteindre cette soutenance est un succès que je partage avec l'équipe d'encadrement. Je remercie donc Bruno, Alexandre et Julien d'avoir partagé leurs connaissances techniques, d'avoir été disponibles pour les réunions pour m'aider à ne pas m'éparpiller, tout en me laissant travailler sur ce qui me plait. Ce n'est pas rien d'avoir réussi à traverser les confinements et multiples autres désagréments de cette pénible pandémie. Et je n'oublie pas l'humour ironique de Bruno, les vastes quantités de saccharose gracieusement offertes par Alexandre, et l'inspiration que représente Julien en matière de barbe. Je remercie également Goran d'avoir chaperonné ma thèse et Elliot pour son accompagnement durant son début.

Cette thèse contient aussi des petits fragments de perspectives de ceux avec qui j'ai discuté de mes divers problèmes. Je remercie notamment Adrien, Baptiste, Benoît, Clara, Eugénie, Hugo, Léonie, Maxime, Othmane, Julien, Juliette et Zoé pour le temps qu'ils m'ont accordé. Restes ceux qui m'ont accompagné pour faire des jeux, boire des bières, échanger des photos d'animaux mignons et autres activités essentielles aux travaux de recherche. Il serait trop long de les citer tous donc j'opte pour un recouvrement fini : merci les camarades de l'ENSTA, tant des études que du labo, merci les collègues l'ONERA, les vieux copains du nord et les nouveaux copains des jeux et des associations.

Et je n'aurais littéralement pas été là sans ma famille. Je remercie en particulier mes grands-parents pour avoir montré l'exemple des grandes études, mes parents pour m'avoir soutenu pendant celles-ci et enfin mon frère Pierre pour m'avoir hébergé pendant la thèse. Sans tout cela j'aurais difficilement pu m'adonner à ma nouvelle passion d'encadrer des fusées dans des losanges.

Enfin, la rédaction de ce manuscrit n'aurait pas été possible sans le miracle ordinaire de la caféine ; j'estime que le texte suivant en contient au moins dix grammes au total, soit une tasse de café par page. A consommer avec modération.

Introduction



Long long long ago, on a distant formless planet lived the Shadoks [1]. They were birds with small wings, long legs they used to pump and a brain that was capable of remembering literally four things but was somehow capable of rocket engineering. They built a house shaped spaceship to migrate to a better planet but their leading scientist professor Shadoko estimated that it had only one chance in a million to fly. To guarantee the safety of their exodus, their strategy was to fail 999 999 launches as fast as they could and embark on the millionth launch, which, according to their understanding of probability, was sure to succeed.

Sadly, this approach is not possible in real life, firstly because the proof of safety is not very convincing and secondly because humanity's industry is not as formidable as the Shadok's, and we cannot afford to crash that many rockets. Hence, the need for Robust Optimal Control, which consists in computing trajectories that simultaneously maximize the chance of success and minimize the overall cost. Its goal is to provide a numerical proof that guarantees ahead of launch that the spaceship will reach orbit even if it suffers some amount of errors and disturbances during the flight,

[1] Les Shadoks, TV series by J. Rouxel, C. Piéplu and J. P. Couturier, 1968.
Image from website maths-et-tiques.fr:
https://www.maths-et-tiques.fr/index.php/detentes/probabilites-chez-les-shadoks

and it will reach orbit using as little fuel as possible. This reduces the loss of fuel and rockets, which could have made the Shadok Space Program cheaper and avoided risking the lives of the Shadoknauts.

During the space race in the 1960s, the question on how to guide a vehicle from point A to point B while using as little fuel as possible gained attention. This lead to the development of Lev Pontryagin's school of control in mathematics [53]. It adapts Hamiltonian theory from mechanics to abstract the control away, which allows to compute an optimal trajectory and its associated control with greater accuracy than with methods that approximate the control directly. That trajectory can be used to guide a launch vehicle, that is, serve as reference trajectory. Only a reference trajectory because the model of the vehicle that was used in its computation is not exact. The discrepancies between model and reality cause the actual trajectory to deviate, requiring a guidance, navigation and control algorithm to correct the trajectory. Then arises the question of how robust such an algorithm is: how much deviation is manageable? Are there scenarios where the controller will fail to complete the mission? Usually, the robustness of the solution is demonstrated by dispersing the parameters around nominal values with Monte Carlo simulations [20] or other probabilistic methods [23], but these methods do not exclude the possibility of failure, which can be problematic on critical systems.

In parallel, Ramon E. Moore published a book on interval arithmetic [47] which birthed modern interval methods in computer science. Interval methods handle intervals that conservatively enclose reals in two floating points instead of approximating reals by a single floating point. This allows to enclose numerical errors: differences between floating point computations and exact computations, as well as method errors: errors that arise from approximations done by a given method. Thus, interval methods provide guaranteed bounds on a result. However, this approach can be too pessimistic: the accumulation of error can cause intervals to grow so much that they are no longer informative, for instance by guaranteeing that a number is between zero and a billion. As a consequence new set representation were proposed to enclose results, notably symbolic zonotopes [29] and constrained zonotopes [63], that compute and store linear correlations between variables and use them to produce tighter enclosures. Methods that use these kinds of sets constitute the school of set-based, or validated methods.

Set-based methods have been applied to a range of problems, including dynamical system and control. A notable tool is validated simulation [45] which can simulate systems with bounded uncertainties, that is a system where the initial state can take an unknown value within a bounded set and which depends on parameters that are also unknown but bounded. This tool can prove the robustness of an autonomous vehicle in respect to errors and disturbances of a given amplitude.

If the control algorithm of an autonomous vehicle can be expressed explicitly, it can be simulated with uncertainties. This can check if the controller leads the vehicle to its destination and if it stays outside unsafe zones [6]. However, they have not been applied to the case where the controller is known implicitly as minimizing a cost, as in the case of a rocket that optimizes its trajectory and control autonomously during the flight as in [16].

To sum up, we have on one hand a school of mathematics that can compute optimal trajectories with great accuracy but lacks robustness, and on the other hand a school of computer science that can enclose sets of trajectory conservatively.

This leads to this thesis, which consists in enclosing sets of optimal trajectories of a problem with uncertainty to guarantee the robustness of the guidance of autonomous vehicles.

Our work is divided in five parts: review of literature; definition of sets of trajectories; computation of those sets, first with intervals then with zonotopes; and applications.

Firstly, Chapter 1 probes the state of the art. The objective is twofold. First, we present the two pillars of this thesis which are optimal control and set-based methods, and list all concepts and tools we use. Notably the first order characterization of optimal trajectories, also known as Pontryagin's Minimum Principle, and validated simulation for models with bounded uncertainty in set-based methods. Then we list trends of robust control and the problematic they answer to, to set objectives for our methods.

Since the literature has not tackled sets of optimal trajectories of autonomous vehicles for problems with uncertainties as we envision, Chapter 2 defines them by combining the first order characterization of optimal trajectories and bounded uncertainties. We propose three sets, one that encloses the trajectory and control of a system that has a perfectly accurate estimation of its parameters and initial state, two sets that enclose trajectories and controls of more concrete systems that suffer from estimation errors and work in open-loop or closed-loop. We then state how these sets can be computed by writing their elements as hybrid dynamical systems with uncertainty that are subject to boundary constraints, and investigating their geometry in the phase diagram.

The goal of Chapter 3 is to put this theory to use, by developing methods that enclose our sets of trajectories using interval methods and validated simulation. Validated simulation encloses dynamical systems with uncertainties while interval methods allow to enclose a solution set from a cloud of solution points. Moreover, interval methods are among the simplest validated methods. However, intervals are also limited: their rectangle shape induces significant over approximation and methods to counteract it are of exponential complexity and this severely reduce the scope of our methods.

In an effort to replace intervals in the methods of Chapter 3 and thus expand their capabilities, Chapter 4 develops new methods for zonotopes.

We first propose constrained symbolic zonotopes by combining the symbolic zonotopes of [29] and the constrained zonotopes of [63]. We notably prove that this set representation can enclose sets defined implicitly as satisfying a constraint by simply computing the constraint function with affine arithmetic and solving linear optimization problems. Then we develop spatio-temporal constrained zonotopes which allow to easily simulate a hybrid system and can be combined with our symbolic zonotopes to solve boundary value problems in a single forward swipe of validated simulation, thus fixing all issues of Chapter 2.

At this point we have methods that can theoretically compute our sets for a wide range of optimal control problems. The goal of Chapter 5 is to challenge them by building a unified protocol that encapsulates the work of all other chapters, then testing it on a sequence of increasingly complex aerospace problems. This highlights the limits of our methods. We nevertheless set out to explore how our results could be used, from classical robustness assessment to exotic methods for control synthesis.

Notations

- Interval variables and interval vectors are enclosed in brackets: $[x]$.
- The bounds of an interval $[a]$ are noted $\underline{a}, \overline{a}$. Its middle point is noted $\text{mid}([a])$.
- Set-valued functions are enclosed in brackets: $[f] : \mathcal{X} \mapsto [f](\mathcal{X})$, even if $\mathcal{X}$ is not an interval.
- $(\cdot)$ such as in $y(\cdot)$ denotes a time function: $y(\cdot) : t \in [t_i, t_f] \mapsto y(t)$.
- $y(\tau)$ and y_τ are different, the later is a vector and is a targeted boundary value of $y(.)$.
- A hat as in $\hat{\xi}$ denotes a variable that is related to an estimation.
- The blackboard bold font denotes a zonotope $\mathbb{Z}$ (except $\mathbb{R}$: it is the set of reals as usual).
- Constrained zonotopes are noted $\mathbb{Z}^{\mathbb{A}}$, where $\mathbb{A}$ is the constraint part.

Chapter 1

State of the Art

In this section, important notions of dynamical systems and notations are introduced. Then, the two pillars of this thesis are presented. First, optimal control with a focus on indirect methods, then validated methods and in particular constrained zonotopes and validated simulation. A summary of related works in robust control are presented in both the optimal control and validated methods sections.

1.1 Dynamical Systems

A dynamical system is a function of time describing the evolution of a vector of variables that represents a system. This system is represented by *Ordinary Differential Equations* (ODE) comprised of a dynamic and an initial point such as in (1.1).

$$\begin{cases} \dot{x}(t) &= f(t, x(t)), \\ x(t_i) &= x_i, \end{cases} \tag{1.1}$$

where $f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is the dynamic, $t_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$ the initial time and initial condition, and $x(\cdot) : \mathbb{R} \to \mathbb{R}^d$ the solution.

The main goal of those mathematical objects is to define the trajectory of a concrete system, and thus find to which future states a given initial state will lead. When the initial state has degrees of freedom, it is convenient to define flow functions (or end-point mappings) as follows: for given initial and final times $t_i, t_f \in \mathbb{R}$, define $\phi_{t_i,t_f} : \mathbb{R}^d \mapsto \mathbb{R}^d$ as (1.2).

$$\phi_{t_i,t_f}(x_i) = x(t_f) \text{ with } x(\cdot) \text{ the solution of } \begin{cases} \dot{x}(t) &= f(t, x(t)), \\ x(t_i) &= x_i. \end{cases} \tag{1.2}$$

A difficulty that arises with nonlinear dynamics is an explosion in finite time, meaning the solution $x(\cdot)$ is only defined on a subset of $\mathbb{R}$.

It is also possible that Equation (1.1) has multiple solutions. Fortunately, Picard-Lindelöf theorem [44, 51] proves the existence and uniqueness of a maximal solution (that is a solution defined on the longest time span possible) if f is locally Lipschitz continuous on x and continuous on t. However, this solution cannot be computed analytically in the general case, which arises the need for numerical approximations. Picard's fixed-point method is one way to compute an approximation, but the more popular approach is to discretize the ODE and simulate it, as with Euler's explicit or implicit methods, Heun's method, or the wide range of other Runge-Kutta methods (see [21, 22]), which are presented in the following section.

1.1.1 Numerical simulation

A first way to approximate the solution of an ODE numerically is directly derived from the proof of Picard-Lindelöf theorem. Consider a sequence of function $x_k(\cdot) : \mathbb{R} \mapsto \mathbb{R}^d$ defined by:

$$\begin{array}{l} \forall k \geq 1, \forall t \in \mathbb{R}, x_{k+1}(t) = x_i + \int_{t_i}^{t} f(s, x_k(s))ds, \\ \forall t \in \mathbb{R}, x_0(t) = x_i. \end{array} \tag{1.3}$$

Fixed-point theorem ensures that $x_k(\cdot)$ converges when $k \to \infty$ to a time function $x(\cdot)$ such that $x(t) = x_i + \int_{t_i}^{t} f(s, x(s))ds$, hence a solution of (1.1). This method can be cumbersome as it requires multiple iterations and an entire trajectory $x_k(\cdot)$ is kept in memory at all time.

Hence the simulation approach, which computes the trajectory in a single forward swipe, is often preferred. The oldest and simplest simulation methods are Euler's (implicit and explicit) methods that approximate the trajectory by a tangent in between time steps. More refined methods exist, notably Taylor's method, which consists in computing the subsequent derivative of the trajectory at time t up to an order n then computing an approximation of $x(t + h)$ using Taylor's theorem (1.4).

$$\begin{array}{rl} x(t+h) & = \sum_{k=0}^{n} \frac{x^{(k)}(t)}{k!} h^k + o(h^n), \\ & = x(t) + \sum_{k=1}^{n} \frac{f^{(k-1)}(t)}{k!} h^k + o(h^n). \end{array} \tag{1.4}$$

This method requires by hand or automatic differentiation, leading many programmers to use Runge-Kutta methods, notably RK4, which is very popular due to its ease of implementation. Indeed RK4 creates a fourth order Taylor approximation of the solution with a weighted average of the first derivative (*i.e.* f) in 4 different locations.

Beside the comparative ease of use, Runge-Kutta methods have the benefit of being more customizable, and can thus better fit a problem. Indeed, Runge-Kutta method of any order with different stability properties and computational resource usage can be implemented, see [22, 37] for details.

Because these simulation methods approximate the integral of the Taylor series of the solution, they require the solution to be sufficiently differentiable. However, many aerospace systems experience brutal change of state and dynamics due to detaching stages or the engine shutting off. This raises the need for a specific class of systems with brutal change of dynamics.

1.1.2 Hybrid systems

Hybrid systems consist of a combination of a discrete automaton and dynamical systems. Due to their multidisciplinary nature and the wide range of systems they model, hybrid systems have multiple formalisms and nomenclatures in the literature. For instance authors stemming from the computer science community such as [9] define a hybrid system as a "discrete program with an analog environment" and endow it with invariants and labeling functions. By contrast, members of the control community define hybrid systems as dynamical systems with "discontinuous dynamics ruled by a partition of the state space" for which the change of discrete variables are not directly controlled [36], and endow it with switch functions and indexed dynamics. Hybrid systems are generally defined as a tuple [9, 10, 33, 36, 52], but the number and nature of attributes vary. Hence, we use the following 4-tuple composed of the 4 recurring attributes, which is the simplest definition required for this thesis:

A hybrid system $\mathcal{H} = \{\mathcal{Q}, \mathcal{M}, \mathcal{F}, \mathcal{S}\}$ is composed of:

- $\mathcal{Q} = \{1..M\}$: a finite set of integers corresponding to discrete automaton modes,
- $\mathcal{M} = \{M_1..M_M\}$: a finite set of continuous state spaces or manifolds, one for each mode,
- $\mathcal{F} = \{f_1..f_M\}$: a finite set of dynamics, one for each mode,
- $\mathcal{S}$: a set of switch maps, each defined by start and finish modes $q^-, q^+ \in \mathcal{Q}$, a switch function $s : M_{q^-} \to \mathbb{R}$ and a jump function $j : M_q^- \to M_{q^+}$.

A switch happens when there is a switch map in $\mathcal{S}$ with start mode $q = q(t)$ and switch function such that $s(y(t)) = 0$. In that case we note t^- the time right before the switch and t^+ right after the switch, and we have: $q(t^-) = q^-, q(t^+) = q^+, y(t^+) = j(y(t^-))$. Note that hybrid system can also be defined by invariants as in [9] or guard sets as in [56]. This is often equivalent as modes could be defined with the invariant $s(x) > 0$ (or $s(x) < 0$) and use set $\{x \in M_{q^-} : s(x) = 0\}$ as guard set. Because Chapter 2 expresses the optimization problem as a boundary value problem, we use switching functions as those turn naturally into boundary constraints.

This formalism encompasses all hybrid phenomena that are highlighted in [48]: state events, change of simulation model, reinitialization, and discontinuous change of the state. The concept of chattering is also highlighted in [48]: it happens when a hybrid system alternates between two modes very quickly. In our context, this is a sign of a faulty model that should be corrected by adding hysteresis or a singular mode. Indeed, a rocket engine cannot be turned on and off repeatedly.

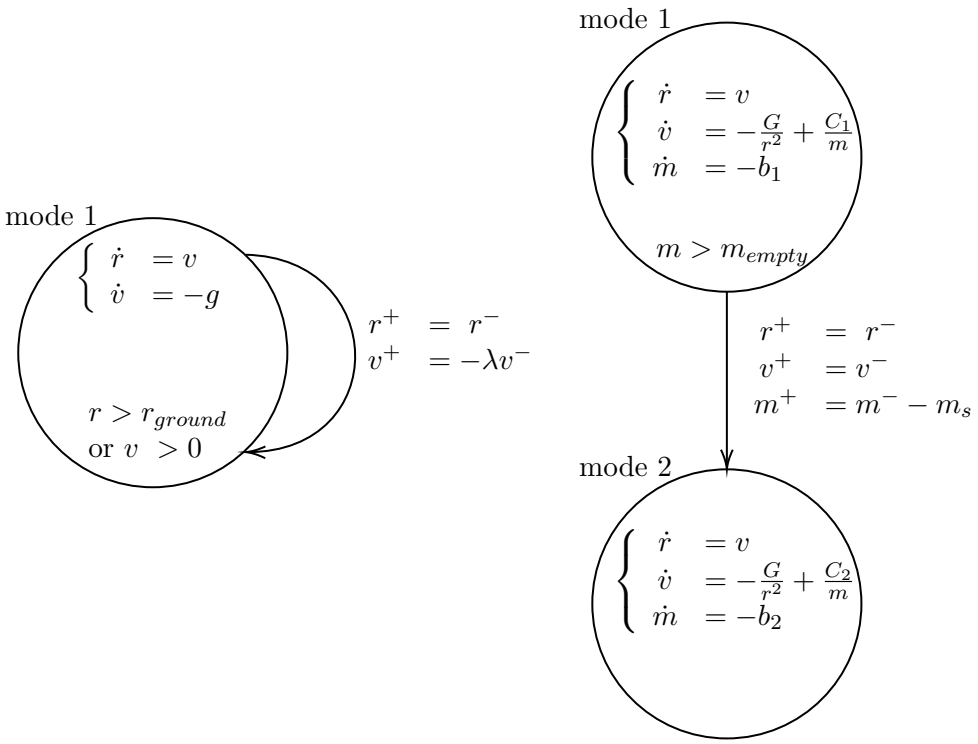


Figure 1.1: Two hybrid automata: bouncing ball (left) and two stage rocket (right)

Two hybrid systems that will be used in future sections are presented in Figure 1.1.

The first system is a bouncing ball, it has one mode: $\mathcal{Q} = \{1\}$, the state space is position and speed $(r, v) \in \mathbb{R}^2$: $\mathcal{M} = \{\mathbb{R}^2\}$, dynamics follow Newton's law with gravity: $F = \{f(r, v) = (v, -g)\}$, and it bounces when it reaches the ground while descending, such that $v^+ = -\lambda v^-$ which translates to a switch from mode 1 to itself: $\mathcal{S} = \{(1, 1, s(r, v) = r - r_{ground} \text{ if } v < 0 \text{ and } s(r, v) = 1, j(r, v) = (0, -\lambda v))\}$.

The second system is a two stage rocket: it has two modes, before and after the first stage is jettisoned:$\mathcal{Q} = \{1, 2\}$, both state spaces are position, speed and mass (r, v, m): $\mathcal{M} = \{\mathbb{R}^3, \mathbb{R}^3\}$,

dynamics follow Newton's law with gravity, thrust and fuel consumption, but with different engine coefficients $F = \{f_1(r,v,m) = (r, -G/r^2 + C_1/m, -b_1), f_2(r,v,m) = (r, -g + C_2/m, -b_2)\}$, and the first stage is dropped when it is empty, leading to switch $\mathcal{S} = \{(1, 2, s(r,v,m) = m - m_{empty}, j(r,v,m) = (r,v,m-m_s))\}$ where m_s is the mass of the empty stage.

Simulation of hybrid systems can be achieved by making simulation timestamps coincide with event timestamps. Indeed, the dynamic of the system are smooth in between events, so simulation methods can be applied. Hence, the main difficulty is detecting events and locating their time of occurrence. Common situations in which an event can be missed entirely are highlighted in [31], notably when the system goes through a guard set then back in between two simulation steps. Once an event is detected, the location of the time of occurrence can be computed with high precision by applying a Newton-like zero finding method to the flow function. To that end, the following section explains how derivatives of the flow can be computed.

1.1.3 Differentiation of the flow

Consider two times t_i and t_f, consider a flow function Φ_{t_i,t_f} as defined in (1.2) and a point x_i. Derivatives of the flow can be computed using the following theorem from [37].

Theorem: *If $\partial f/\partial x$ exists and is differentiable along the trajectory starting in x_i, then the flow function is differentiable and the following holds: $\partial\Phi_{t_i,t_f}/\partial x_i = \mathbf{R}_{t_i,t_f}(x_i)$, $\partial\Phi_{t_i,t_f}/\partial t_i = \mathbf{R}_{t_i,t_f}(x_i)\cdot f(t_i,x_i)$, where $\mathbf{R}_{t_i,t_f}(x_i)$ is the **resolvent** matrix defined by system* (1.5)*, and $\partial\Phi_{t_i,t_f}/\partial t_f = f(t_f, x(t_f))$.*

$$\begin{cases} \dot{x}(t) & = f(t,x), \\ \dot{\mathbf{R}}_{t_i,t}(x_i) & = \dfrac{\partial f}{\partial x}(t,x(t)) \cdot \mathbf{R}_{t_i,t}(x_i), \\ x(t_i) & = x_i, \\ \mathbf{R}_{t_i,t_i}(x_i) & = I_d. \end{cases} \tag{1.5}$$

A similar formula exists to compute the derivative of the flow with respect to parameters [55], it is used for local sensitivity analysis. Indeed, sensitivity analysis evaluates how sensitive a system is to a variation of parameter, and the resolvent maps small perturbations of the initial state and parameters to subsequent deviation.

Resolvents can be used to define and differentiate the flow of a hybrid system. The flow of a hybrid systems can be defined by composition of flows and jumps. As an example, consider the two stage launcher of Figure 1.1. It starts in Mode 1 with dynamic f_1 until switch time t_1 such that $s(t_1, x(t_1)) = 0$, is transported to the space of Mode 2 by jump function j and continues with dynamic f_2 until final time t_f.

This leads to formula $\Phi_{t_i,t_f}(x_i) = \Phi^2_{t_1,t_f}(j(\Phi^1_{t_i,t_1}(x_i)))$, with $\Phi^2_{t_1,t_f}$ the flow of dynamics f_2 and $\Phi^1_{t_i,t_1}$ the flow of dynamics f_1.

However, differentiating this flow is difficult because a change of initial state results in change of switch time:

1. start with a small perturbation to the initial state $x(t_i) = x_i + \delta x_i$,
2. this changes the intermediate state $x(t_1) = x_1 + \delta x_1$ with $\delta x_1 \approx \mathbf{R}^1_{t_i,t_1} \cdot \delta x_i$,
3. which changes the value of the switch function $s(x(t_1)) = \delta s$ with $\delta s \approx \nabla s \cdot \delta x_1$,
4. as a consequence the switch time changes by δt_1 such that $0 = s(x(t_1 + \delta t_1)) \approx \nabla s \cdot f_1 \delta t_1 + \delta s$.

Considering t_1 as a function of x_1 leads to a cumbersome formula. Hence, it is often more convenient to consider switch times as degrees of freedom for the purpose of flow differentiation. This means defining a flow that takes switch times as inputs: $\Phi_{t_i,t_f}(x_i, t_1)$ and differentiating it by chain rule. However, the dependencies between initial state and switch times will need to be taken into account by the program at another level.

Note that there are instances where the derivative of the flow does not exist, notably when the system rides along a guard set such that a small perturbation will change the structure of the trajectory. In our applications we take systems for which we know in advance that they go through a sequence of modes in order, so we can safely assume that this does not happen.

All the tools that were presented in this section describe systems that are determined by their initial state and dynamics. However, controlled systems have an additional input that is chosen online. Next section discusses how to model such systems.

1.2 Optimal control

A controlled system is a time varying system that takes another input chosen by a controller. It takes the form (1.6)

$$\begin{cases} \dot{y}(t) &= f(t, y(t), u(t)), \\ y(t_i) &= y_i, \end{cases} \tag{1.6}$$

where $y(\cdot)$ is the state trajectory and $u(\cdot) : [t_i, t_f] \mapsto \mathbb{R}^{d_u}$ is the control. This control is chosen with respect to certain goals, like reaching a desired end point without entering unsafe regions.

Such specifications often leave degrees of freedom, which can be chosen so as to minimize a cost. This yields an Optimal Control Problem (OCP) such as (1.7).

$$\begin{array}{ll} \min\limits_{u(\cdot):[t_i,t_f]\to\mathcal{U}} & \int_{t_i}^{t_f} \ell(t,y(t),u(t))dt \\ \text{s.t.} & \left\{ \begin{array}{ll} \dot{y}(t) & = f(y(t),u(t)), \\ y(t_i) & = y_i, c_f(t_f,y(t_f)) = 0, t_f \text{ is free.} \end{array} \right. \end{array} \tag{1.7}$$

Interpretation We are searching a final time $t_f \in \mathbb{R}$, a control $u(\cdot) : [t_i, t_f] \to \mathcal{U} \subset \mathbb{R}^{d_u}$ and a corresponding trajectory $y(\cdot) : [t_i, t_f] \to \mathbb{R}^d$ such that the system goes from an initial state $y_i \in \mathbb{R}^d$ to a final state that satisfies a constraint $c_(t_f, y(t_f)) = 0$, with $c_f : \mathbb{R}^{d+1} \to \mathbb{R}^m$, while minimizing the integral of an instantaneous cost $\ell : \mathbb{R} \times \mathbb{R}^d \times \mathcal{U} \to \mathbb{R}$.

The first question that arises is whether this problem has a solution. The topic of the existence of a solution is vast, hence this thesis will only use the following existence theorem from [65].

Theorem *Assume that $\mathcal{U}$ is compact, that there may be state constraints $c_0(y) \leq 0, c_1(y) \leq 0 \cdots$, where $c_0, c_1 \cdots$ are continuous functions on $\mathbb{R}^d$, that $\mathcal{Y}_f$ is reachable from y_i, that every trajectory steering the system from y_i to $\mathcal{Y}_f$ is bounded by a same constant and that the set $\{(\ell(t,y,u) + \gamma, f(t,y,u)) | u \in \mathcal{U}, \gamma \in \mathbb{R}^+\}$ is convex for all $t \in \mathbb{R}$ and $y \in \mathbb{R}^d$.*

Then there exists an optimal control such that the corresponding trajectory steers the system from y_i to $\mathcal{Y}_f$.

When the solution does exist, it can be approximated numerically. The main difficulty is that $y(\cdot)$ and $u(\cdot)$ are time functions, hence variables of infinite dimension. Many methods exist and they mainly fall in two categories:

- *direct methods*: the control is discretized, thus turning the infinite dimensional problem into an Euclidean nonlinear optimization problem such as (1.8).

- *indirect methods*: using a characterization of the optimal trajectory and control, the infinite dimensional optimization problem is transformed into a Boundary Value Problem (BVP).

Direct methods suffer from a lack of precision due to the discretization but they require little prior analysis and no knowledge of the structure of the optimal trajectory. Some methods can be found in [15], though the direct method field is as vast as Euclidean optimization.

$$\begin{array}{ll} \min\limits_{(u_n)\in\mathcal{U}^N} & \sum_{n=0}^{N} \ell(t_n, y_n, u_n)dt \\ \text{s.t.} & \left\{ \begin{array}{ll} y_{n+1} & = \Phi_{t_n,t_{n+1}}(y_n, u_n), \\ y_0 & = y_i, y_N \in \mathcal{Y}_f, t_N \text{ is free.} \end{array} \right. \end{array} \tag{1.8}$$

Direct method are particularly fit when the discretization rate can be set to the controller's frequency as it makes the control representation exact. For instance, a popular school of control that is very similar if not part of direct control is Model Predictive Control (MPC) [60]. It is an online closed-loop, or feedback control technique that uses a model of the dynamic to compute the optimal sequence of controller inputs, applies the first input then solve the problem again at the next time step with new sensor data. The main difference with the problem at hand is that historically, MPC focuses on linear and quadratic models that can be solved online and has a receding horizon since it applies to ongoing systems such as batch processes, though MPC has grown to encompass a lot more problems [34, 54].

In aerospace design for example, controllers cannot run an expensive optimization method due to severe limitation in computational resource. Instead, they are designed to make the launch vehicle follow a reference trajectory that is planned offline, that is before the launch. This reference trajectory needs to be computed with high precision, which is where indirect methods shine. The main approach is to use the first order optimality condition, also known as Pontryagin's Minimum Principle (PMP). It transforms the OCP into a Euclidean two point boundary value problem [19, 65]. This problem may be solved with a shooting method, an algorithm that takes an initial guess, simulates it until the end time, measures how far the systems lands from the desired final set, then corrects the initial guess using a Newton-like algorithm, and repeats. Such algorithms have a high local convergence rate leading to higher precision than direct methods when they do converge. However, they will not converge if they are initialized too far from the solution.

To solve the initial guess issue, shooting methods can be embedded in a continuation (or homotopy) method [5]. This procedure uses the fact that if an OCP is deformed continuously, the optimal solution is deformed continuously as well. It starts by solving a simplified version of the problem, then make the problem slightly closer to the actual problem, solves it again using the preceding solution as initial guess, then repeats until the actual problem is solved. We refer the reader to [15] for a more thorough presentation and to [20] for an implementation for aerospace system for where the continuation is done by slowly changing parameter values.

In order to tackle a wide range of problems, the PMP is generalized. A formulation for hybrid systems can be found in [30, 33, 36],

and a formulation of the PMP for a discrete set of trajectories can be found in [18]. The hardest generalization is when the OCP is subject to state constraints, *e.g.* when the speed of a vehicle cannot exceed a dangerous threshold. A PMP with state constraints is stated in [26], but it replaces trajectories with measures, which makes the problem significantly more challenging. As a consequence further analysis of the state constraints is required to implement numerical methods. A survey in optimal control with state constraints can be found in [38, 67].

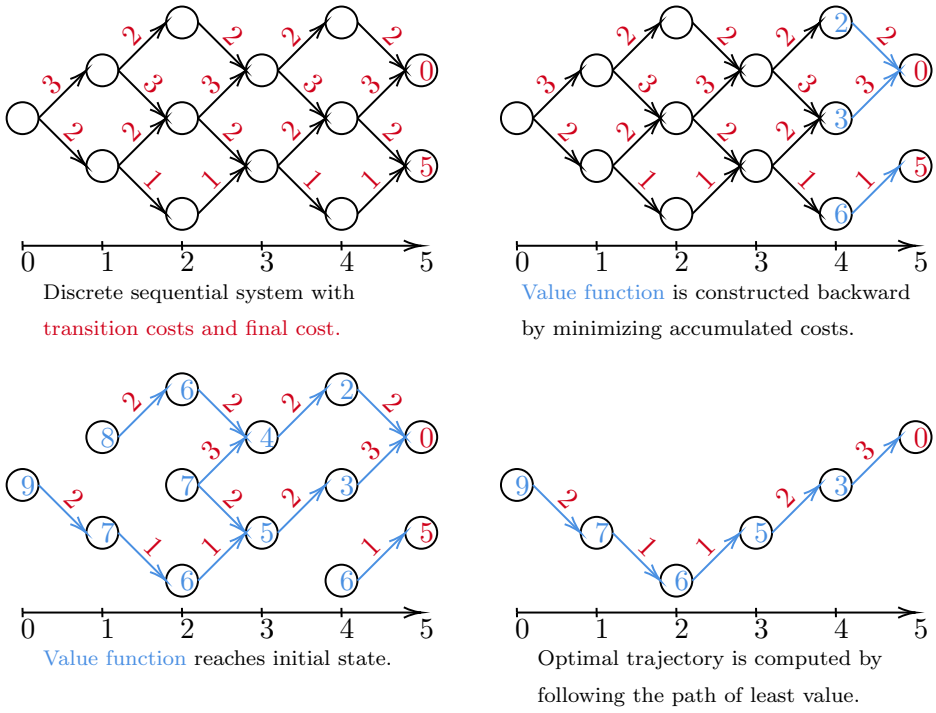


Figure 1.2: A simple example of dynamic programming.

A drawback of both direct and indirect methods is that they are local, they have no guarantee to converge to a global minimum. In fact some can even converge to a local maximum as first order optimality conditions are the same for all extremums. This arises the need for second order conditions as can be found in [46]. The main global method for OCPs is based on Bellman's dynamic programming principle [12]. This principle consists in cutting an optimization problem into smaller more manageable problems and linking them together with a value function, it is illustrated in Figure 1.2. The application of this principle to an OCP yields the Hamilton Jacobi Bellman (HJB) partial differential equation [41], which, when solved, allows to find the globally optimal trajectory by following the path of least value.

However, the HJB approach requires the computation of a value field across all state space, which implies a numerical complexity that is exponential in the dimension and limits it to low dimension problems. Nevertheless, the existence of the value field has ramifications with the PMP: under some assumptions the co-state in the PMP is the gradient of the HJB value function along the optimal trajectory [27]. We believe it is a nice way to make sense of the otherwise abstract theory presented in the next section.

1.2.1 Pontryagin's Principle

In this section, the indirect approach is presented in greater detail. It addresses OCP (1.7) by introducing a co-state $p(\cdot) : [t_i, t_f] \to \mathbb{R}^d$ (also known as adjoint vector) to characterize the optimal solution with Principle (1.9).

Pontryagin's Minimum Principle (PMP) If $(y(\cdot), u(\cdot))$ are a normal optimum, then there exists a nontrivial co-state $p(\cdot)$ such that $(y(\cdot), p(\cdot), u(\cdot))$ satisfy the following equations:

$$\begin{array}{l} H(t, y, p, u) = \ell(t, y, u) + p.f(t, y, u) \\ \dot{p}(t) = -\dfrac{\partial H}{\partial y}(t, y(t), p(t), u(t)) \\ \forall t \in [t_i, t_f], u(t) \in \arg\min_{v \in \mathcal{U}} H(t, y(t), p(t), v), \\ C_f(t_f, y(t_f), p(t_f)) = 0 \end{array} \tag{1.9}$$

where H is the pre-Hamiltonian and $C_f : \mathbb{R}^{1+2d} \to \mathbb{R}^d$ is an end point constraint function that depends on the constraints and boundary cost of the initial problem (see [19]). The optimal control $u(\cdot)$ is implicitly defined as minimizing the pre-Hamiltonian at every time.

In many control problems including Goddard's problem [19], which is the aerospace problem we tackle, this implicit definition yields an explicit expression. That is, there is a function $\mu : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{U}$ such that:

$$u \in \arg\min_{v \in \mathcal{U}} H(y, p, v) \iff u = \mu(y, p). \tag{1.10}$$

The equations of the PMP (1.9) can be combined with those of Problem (1.7) to get uncontrolled System (1.11).

$$\left\{ \begin{array}{ll} \dot{y}(t) & = f(y, \mu(y, p)) \\ \dot{p}(t) & = -\frac{\partial H}{\partial y}(y, \mu(y, p), p) \\ y(t_i) = y_i, & \\ C_f(t_f, y(t_f), p(t_f)) = 0. & \end{array} \right. \tag{1.11}$$

Equation (1.11) is a two point boundary value problem where one searches an initial co-state p_i such that the end point of Dynamics (1.11) satisfy $C_f(t_f, y(t_f), p(t_f)) = 0$.

See [19, 20, 64, 65] for numerical shooting methods that solve this problem with aerospace systems.

The theory of optimal control appeared in the 1960s and the first optimal control based controllers produced disappointing results, notably crashing a plane in a simulator [62]. As a consequence later methods focused on making optimal control more robust.

1.2.2 Stochastic methods for robust optimal control

The failings of optimal control based controllers is caused by the imperfection of the model. This is exacerbated by the fact that optimal control tend to incentivize risky trajectories, like riding along the boundary of an unsafe set. Indeed, the shortest path around an obstacle often skims its surface. As a consequence much effort has been done to make trajectories more robust.

A common approach consists in simulating the system with parameter values taken at random around the nominal value [20], similarly to a Monte Carlo method. More developed stochastic methods combine those random trajectories with Gaussian kernels to better approximate the probabilistic field of trajectories and can thus solve a chance constrained optimal control problem in order to guarantee a probability of success [23]. However, stochastic methods cannot completely exclude the possibility of failure which can be problematic on critical systems.

In parallel, a school of computer science was developed with the precise objective of completely excluding all possibility of errors.

1.3 Set-based methods

This section introduces the second pillar of this thesis: set-based, or validated, methods. Those consists in enclosing reals in sets rather than approximating them with floating points numbers.

To manipulate these sets, real valued function are overloaded with conservative set valued counterparts. That is, for any $f : x \mapsto f(x)$, one can create an inclusion function $[f] : [x] \mapsto [f]([x])$ satisfying Inclusion (1.12).

$$[f]([x]) \supseteq \{f(x) | \forall x \in [x]\}. \tag{1.12}$$

This means that outputs are sets that are guaranteed to contain the actual result.

Validated methods started with interval arithmetic that was popularized by Moore [47] then Neumaier [49].

1.3.1 Set representation using interval arithmetic

An interval $[x]$ is a convex subset of $\mathbb{R}$ that contains all reals between a lower bound $\underline{x}$ and an upper bound $\overline{x}$, with $\underline{x}, \overline{x} \in \mathbb{R} \cup \{-\infty, +\infty\}$. Interval arithmetic was proposed as an alternative to floating-point arithmetic to avoid errors due to numerical errors [47] but they have gained more use cases.

Interval vectors, or *boxes*, are an axis-aligned rectangular set in a finite dimensional space. They are an inexpensive representation of a high dimensional set (compared to polytopes) but may induce a wrapping effect during computations [40].

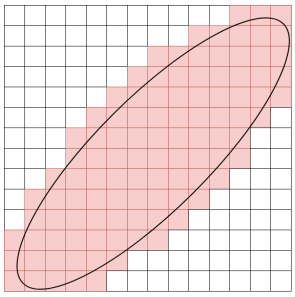


Figure 1.3: The paving approach to enclosing sets. The search space is cut into mutually disjoint boxes and the red boxes that intersect with the ellipsoidal set form an outer enclosure of that set.

In lower dimension, an accurate enclosure of a set can be achieved with a paving of boxes, a set of mutually disjoint boxes which together cover the whole set. A simple paving is illustrated in Figure 1.3. Such a representation is potentially very precise, but the computational cost grows very fast in high dimension. Indeed, to double the precision, the number of tiles has to be doubled in each direction, which means 2^n as many tiles, where n is the dimension of the vector space. Hence, the complexity of this representation is exponential in the dimension of the state, which makes it ill-suited for many practical cases.

As a consequence, solvers with polynomial complexity were developed, notably by using contractors. Contractors are interval functions used to enclose the solutions of a set Constraint Satisfaction Problem (CSP) of the form:

$$\text{find all } x \text{ such that } 0 \in [A] \cdot x + [b] \text{ or more generally } 0 \in [f](x).$$

A contractor $\mathcal{C}$ takes a box $[x]$ as input and outputs a possibly smaller box that contains all possible solutions in $[x]$, that is $\mathcal{C}([x])$ is such that $\{x \in [x] : 0 \in [f](x)\} \subset \mathcal{C}([x])$.

Contractors can be defined as interval counterparts of numerical algorithm, for instance Gauss elimination and Gauss-Seidel Algorithm for $0 \in [A] \cdot [x] + [b]$ and Newton's method for $0 \in [f](x)$. Alternatively they can be built using interval constraint propagation techniques, yielding the forward backward contractor, or linear programming. A list of those contractors and their workings can be found in [40].

Later sections will use Krawczyk's contractor $\mathcal{C}_k$, which is inspired by Newton's method to enclose $0 \in [f](x)$ and can be defined as follows [40]:

Let $\text{mid}([x])$ the middle point of $[x]$, $\mathcal{C}_K : \mathbb{IR}^d \to \mathbb{IR}^d$ ($\mathbb{IR}$ is the set of real intervals) is defined as follows:

$$\mathcal{C}_K([x]) = [x] \cap \left(\begin{array}{c} \text{mid}([x]) - M \cdot f(\text{mid}([x])) + \\ \left(I_d - M \cdot \left[\frac{\partial f}{\partial x}\right]([x])\right) \cdot ([x] - \text{mid}([x])) \end{array} \right), \quad (1.13)$$

where $[\partial f/\partial x]([x])$, an enclosure of the first derivative of f on $[x]$ in the form of an interval matrix and M is an invertible real matrix, typically $M = \text{mid}([\partial f/\partial x]([x]))^{-1}$.

Contractors are generally embedded in iterative solvers, firstly because doing multiple contractions $[x]_{k+1} = \mathcal{C}([x]_k)$ will yield tighter enclosures on most problems, secondly because such procedure requires that $[x]_0$ be an a priori enclosure of all solutions, as all solutions outside of $[x]_0$ are ignored. When no bounds are known and when inputting $[x]_0 = [-\infty, +\infty]$ does not work, an enclosure of a connected set of solutions can be built using an inflate procedure[40]. i) Initialize $[x]_0$ with a search area containing at least one solution (which can be computed by solving the problem with the center point of $[f]$ with a numerical solver). ii) Repeat $[x]_{k+1} = [\S]_{\|} + (1+\lambda) * ([x]_k - \text{mid}([x]_k))$ until $\mathcal{C}([x]_k)$ is in the interior of $[x]_k$, that is, until $\mathcal{C}([x]_k)$ does touch the boundaries of $[x]_k$. Indeed, if there are locally connected solutions outside $[x_k]$, then there would be a path of solutions that would go through the boundary of $[x]_k$ and $\mathcal{C}([x]_k)$ would enclose that path, hence touch the boundary, hence not be in the interior. This is a simple way to enclose a set from a single point, however it is not guaranteed to terminate, since the set could be unbounded, or over-approximation may make it appear so.

Over-approximation is indeed a prominent problem of boxes: because of their rectangular axis-aligned shape, they tend to enclose sets loosely. For instance the two-dimensional needle shape set $\{(x, x) | x \in [-1, 1]\}$ would be represented by the box $[-1, 1] \times [-1, 1]$, thus losing all correlation between the two variables. This over-approximation is known as the wrapping effect, and it can snowball with computations to a point where boxes explode in width and are no longer informative. As a consequence, intervals are ill fitted for many problems and other set representations were developed, notably zonotopes [29].

1.3.2 Zonotopes and constrained zonotopes

Zonotopes are subsets of $\mathbb{R}^d$ defined by the affine deformation of a box of dimension $d_\epsilon \geq d$, as showcased in Figure 1.4.

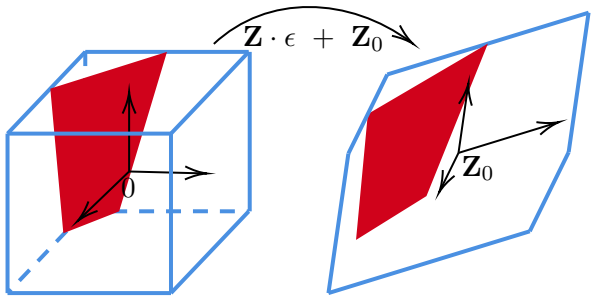


Figure 1.4: Zonotope and constrained zonotope. Left: The noise box $[-1, 1]^{d_\epsilon}$ (blue) and the intersection of the unit box with constrained set $\mathbf{A} \cdot \epsilon + \mathbf{A}_0 = 0$ (red). Right the d dimensional deformation of these sets by $\mathbf{Z} \cdot \epsilon + \mathbf{Z}_0$ with $d = 2$. Zonotope (blue) and constrained zonotope (red).

A zonotope $\mathbb{Z} = [\mathbf{Z}, \mathbf{Z}_0]$ is defined by $\left\{\mathbf{Z} \cdot \epsilon + \mathbf{Z}_0 : \epsilon \in [-1, 1]^{d_\epsilon}\right\}$, where $\mathbf{Z}$ is the generator matrix, of dimension $d \times d_\epsilon$ and $\mathbf{Z}_0$ is the center vector of dimension $d \times 1$. A zonotope is represented in blue right of Figure 1.4.

There are multiple zonotope formalisms that we separate in two categories: symbolic zonotopes [29] and the others. Symbolic zonotopes retains dependencies between distinct zonotopes by having each noise symbol, textite.g. each coordinate ϵ_i of ϵ have a unique identifier that is shared across all zonotopes, thus connecting the i^{th} column of every generator matrices. New noise symbols are created during computations to enclose errors and nonlinear phenomena to satisfy the Fundamental Invariant of Affine Arithmetic [29], which states: *"at any stable instant in an affine arithmetic computation, there is a single assignment of values from* $[-1, 1]$ *to each of the noise variables in use at the time that makes the value of every affine form equal to the value of the corresponding quantity in the ideal computation"*[29]. Retaining linear correlations between all zonotopes in a program is useful, notably for distributed systems [28], however it requires a unique noise symbol provider and forces the program to encode zonotopes as sparse, or column-wise sparse matrices, since d_ϵ is proportional to the total number of operations in the program and most columns are full of zeros.

On the other hand, non-symbolic zonotopes as in 1.3.2 can be encoded as plain matrices but retain no correlation between separate zonotopes.

Zonotopes are popular because they represent affine operations very well and the computational cost is customizable. Indeed, the computational cost of a zonotope depends on the number of generators, that is the number of nonzero columns in $\mathbf{Z}$.

To achieve a good precision/computational cost trade-off, the number of generators is reduced through operations called reductions, which encloses a zonotope in a bigger zonotopes with fewer generators.

One shortcoming of zonotopes however is that they are not closed under intersections. This leads to the development of constrained zonotopes.

We refer specifically to constrained zonotopes as proposed in [63]: a constrained zonotope $\mathbb{Z}^{\mathbb{A}} = [\mathbf{Z}, \mathbf{Z}_0, \mathbf{A}, \mathbf{A}_0]$ is the affine deformation of a unit box intersected with a plane $\left\{\mathbf{Z} \cdot \epsilon + \mathbf{Z}_0 : \epsilon \in [-1, 1]^{d_\epsilon}, \mathbf{A} \cdot \epsilon + \mathbf{A}_0 = 0\right\}$, where $\mathbf{A}$ the constraint matrix and $\mathbf{A}_0$ the constraint vector. See red in Figure 1.4. We propose our own notations because we consider a zonotope $\mathbb{Z}$ and its constraint part $\mathbb{A}$ as two distinct zonotopes, while they are considered a single entity in [63]. This will be explained in greater detail in Chapter 4.

They are practical because they benefit from the efficiency of affine arithmetic while being closed under intersection with themselves and hyperplanes.

Their semi implicit form is hard to represent exactly but an over approximation can be built with the procedure showcased in Figure 1.5. Indeed, a constrained zonotope $\mathbb{Z}^{\mathbb{A}}$ can be bound in a direction $\mathbf{d}$ by solving the Linear Problem (LP):

$$\begin{aligned} &\max && \mathbf{d}^T \cdot (\mathbf{Z}_0 + \mathbf{Z} \cdot \epsilon). \\ &\epsilon \in [-1, 1]^{d_\epsilon} \\ &\mathbf{A}_0 + \mathbf{A} \cdot \epsilon = 0 \end{aligned}$$

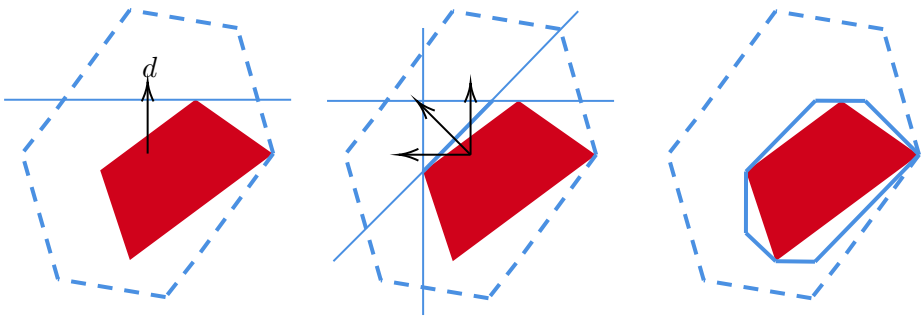


Figure 1.5: Enclosing a Constrained Zonotope. i) Choose a direction d and bound the constrained zonotope in this direction by solving a LP. ii) Repeat for several directions. iii) Build an outer enclosure.

Enclosing a constrained zonotope requires the solution of a LP, hence the need for a validated linear solver, like the one proposed in [50].

Zonotopes and intervals can be used to enclose trajectories of dynamical systems.

1.3.3 Validated simulation

Validated simulation can simulate a set membership ODE like:

$$\begin{cases} \dot{x}(t) & \in [f](x,t) \\ x(t_i) & \in [x_i]. \end{cases}$$

Such an ODE has a set of possible states and a set of possible dynamics at every time. As a consequence, its solution is a tube of possible trajectories rather than a single trajectory, see orange in Figure 1.6. Validated simulation encloses every possible trajectories in a sequence of sets, in our case zonotopes and boxes.

To that end, the time range $[t_i, t_f]$ is discretized in $(t_n)_{n\in 0..N}$, $t_0 = t_i, t_N = t_f$. Starting with an enclosure $[x_n]$ of the systems at time t_n (which is either the input $[x_i]$ or the output of a previous state) an enclosure of the system on the whole time range $[t_n, t_{n+1}]$ (called a Picard box) is built. This is done by computing a box $[\tilde{x}]$ that is a fixed point of interval valued Picard's Operator (1.3):

$$[x_n] + [0, t_{n+1} - t_n][f]([t_n, t_{n+1}], [\tilde{x}]) \subset [\tilde{x}]$$

by fixed point iterations. Indeed:

$$\begin{aligned} \forall k \geq 1, \forall t \in [t_n, t_{n+1}], x(t) &= x_n + \textstyle\int_{t_n}^{t} f(s, x(s))ds, \\ &\in [x_n] + \textstyle\int_{t_n}^{t} [f](s, [\tilde{x}])ds, \\ &\in [x_n] + [0, t_{n+1} - t_n][f]([t_n, t_{n+1}], [\tilde{x}]), \end{aligned} \tag{1.14}$$

Then this Picard box is used to compute an enclosure of $x(t_{n+1})$. In [4], interval Runge-Kutta method are used coupled with inflating terms that enclose the truncation error of the method. Indeed, if the dynamics are sufficiently differentiable, the truncation error can be bounded by evaluating the Lagrange remainder of the difference between the Taylor series of the actual solution and the Taylor series of the Runge-Kutta approximation. For instance, if dynamics are four time differentiable, the truncation error of Runge-Kutta 4 may be enclosed.

The output of validated simulation is showcased in Figure 1.6

Validated simulation is akin to simulating multiple systems between two common time stamps. A time switch is a time horizon shared by all systems, hence all systems can be simulated by doing a first simulation until the switch and another simulation starting right after the switch. Contrarily, a variable time horizons or a state dependent transitions differs from one system to another. As a consequence, there is no shared time stamp to use as a duration for the simulation, which makes validated simulation challenging.

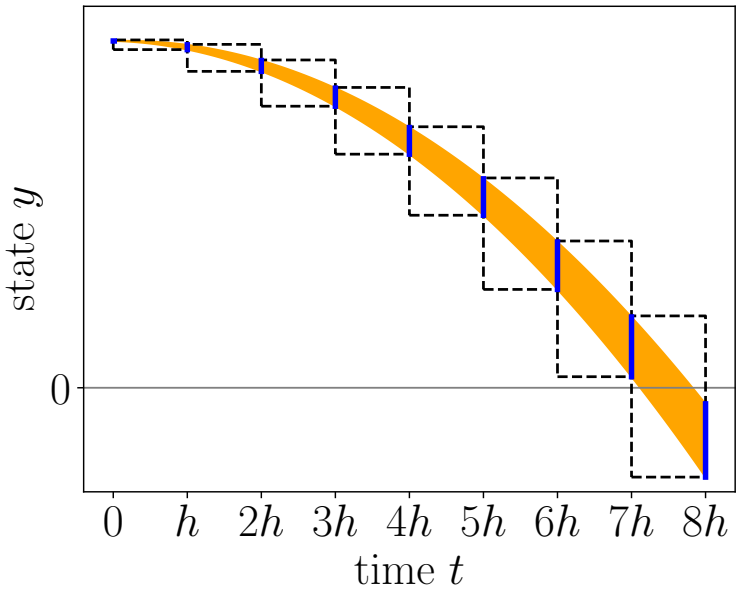


Figure 1.6: Possible trajectories of a falling ball with unknown initial velocity (orange) enclosed with validated simulation library DynIbex [4]. The propagation of the uncertainty on the initial velocity creates a tube of possible trajectories. Validated simulation encloses trajectories in zonotopes (blue, plain) at the endpoints of each time range, and by Picard boxes (black, dashed) on time ranges.

There are several validated simulation libraries that enclose dynamical or hybrid systems with uncertainties in the literature:

- The Matlab toolbox COntinuous Reachability Analyzer(CORA) [7]. It represents sets as zonotopes (or polynomial zonotopes, where noise symbols are multiplied with one another) and simulates nonlinear dynamics with Taylor expansions. It simulates hybrid systems by converting to other set representations during events.

- The software FLOW* [25]. It represents sets as Taylor models: polynomials that coincide with the Taylor expansion of the actual state, with an interval remainder term to enclose truncation errors. These Taylor models can be multiplied with the Taylor expansion of nonlinear dynamics using adapted arithmetic. It simulates hybrid systems by contracting the state set during event using interval contractors, and converting to other set representations [24].

- The C++ toolbox Computer Assisted Proofs in Dynamics (CAPD) [42]. It represents sets as parallelepipeds and simulates dynamics using Taylor expansions.

- SpaceEx [32]. It represents sets as polyhedra in support function representation, which consists in bounding the set in multiple directions similarly to Figure 1.5. It focuses on affine dynamics and applies specialized techniques to compute the evolution support functions.

- The C++ toolbox DynIbex [4]. It represents sets as symbolic zonotopes and simulates nonlinear dynamics using validated Runge-Kutta methods.

- The Julia toolbox JuliaReach [17]. It represents sets as ellipsoids, boxes, polytopes or zonotopes. It does not have an embedded algorithm to simulate hybrid systems but provides a framework to implement one.

The reader may refer to [7] for more libraries.

We use DynIbex [4], first because it is developed by our laboratory, and second because Chapter 4 develops novel uses of symbolic zonotopes to simulate hybrid systems and enclose BVPs.

Such methods can be used to assess robustness.

1.3.4 Set-based methods for robust control

A notable field of set-based robust control is viability theory [11] that computes the set of positions from which there exist an admissible control and can thus deduce safe sets.

Similarly, reachability analysis as in [6, 35, 56] as well as in previously cited validated simulation libraries [4, 7, 25, 32, 42], computes the set of position that can be reached from a given initial position for a given set of possible inputs. Reachability analysis can answer two questions: either the input is a control in which case it assesses whether the vehicle can reach its target, or the input is a disturbance in which case it assesses whether the system can be nudged in an unsafe set.

Interval methods can also be used to prove stability, for instance by computing the invariant set of a system [61].

Lastly, validated method can be used for worst case control synthesis: [57, 58] propose algorithms to compute a control that stirs a system to a desired state for any realization of a bounded noise while achieving the lowest upper bound on the cost.

However, we set out to tackle a slightly different setting in which the control is not chosen by the user but rather computed by an autonomous controller that solves an OCP, like the rocket does before the descending phase in [16]. In this case the control is implicitly defined as solution of an optimization problem, and since we did not find a definition for such sets of trajectories in the literature, so we propose our own in the following chapter.

Chapter 2

Characterization of sets of optimal trajectories for systems with uncertainties

In this section, the main matter of this thesis is presented. Our goal is to build conservative enclosures of optimal trajectories of an OCP subject to uncertainties, so as to quantify the robustness of the optimal control approach in respect to quantitative difference between the model and reality. The motivation is to assess whether a given OCP outputs an adequate trajectory for any realizations of the initial state and parameter functions. Indeed, a trajectory is characterized by two realizations: the actual trajectory and parameter function $y(\cdot), \xi(\cdot)$ and the trajectory and parameter function estimated by the system $\hat{y}(\cdot), \hat{\xi}(\cdot)$. Both parameter variables are taken amidst the same bound $[\xi]$ but they may differ as the actual parameters are often unknown in practice. Trajectories differ notably when the initial state is not estimated $\hat{y}(0) \neq y(0)$ or due to difference between the actual dynamics and the model.

Hence, the following three enclosures are proposed:

- An *anticipative enclosure* containing trajectories whose control is computed with the perfect knowledge of the parameter function $\hat{\xi}(\cdot) = \xi(\cdot)$ and trajectory $\hat{y}(\cdot) = y(\cdot)$
- An *open-loop enclosure* containing trajectories whose control is computed once at the start with an inaccurate parameter function $\hat{\xi}(\cdot) \neq \xi(\cdot)$ and initial state $\hat{y}(0) \neq y(0)$ and then followed blindly.
- A *closed-loop enclosure* containing trajectories whose control is computed several times online with accurate measurements of the state but an inaccurate parameter function $\hat{\xi}(\cdot) \neq \xi(\cdot)$.

To create those enclosures, we state the problem, then make three sets of hypotheses, deduce analytical and geometrical properties from these hypotheses and illustrate them on an example.

2.1 Problem statement

In this section, the main problem is formally stated, as well as assumptions on the uncertainties.

2.1.1 OCP with uncertainties

Consider the following Optimal Control Problem with uncertainties, inspired by both optimal control and robust control:

$$\begin{array}{l} \min\limits_{u(\cdot)} \displaystyle\int_{t_i}^{t_f} \ell(y(t),u(t),\xi(t))dt + \Psi(t_f,y(t_f)), \\ \text{s.t.} \left\{ \begin{array}{l} \dot{y}(t) = f(y(t),u(t),\xi(t)), \\ y(t_i) \in \mathcal{Y}_i, \quad c_f(t_f,y(t_f)) = 0, \quad t_f \text{ is free}, \\ c(y(t)) \leq 0. \end{array} \right. \end{array} \tag{2.1}$$

This problem is characterized by the following data:

- a state $y(t) \in \mathbb{R}^d$, the time function $y(\cdot) : [t_i, t_f] \to \mathbb{R}^d$ is the trajectory,
- a control input $u(\cdot) : [t_i, t_f] \to \mathcal{U}$,
- parameters $\xi(t) \in [\xi]$. They are either constant or time dependent disturbances $\xi(\cdot)$, *e.g.* estimation errors, material fatigue,
- dynamics $f : \mathbb{R}^d \times \mathcal{U} \times [\xi] \to \mathbb{R}^d$,
- an instantaneous cost $\ell : \mathbb{R}^d \times \mathcal{U} \times [\xi] \to \mathbb{R}$,
- a final cost $\Psi : \mathbb{R}^{d+1} \to \mathbb{R}$,
- an initial state $y_i \in \mathcal{Y}_i \subset \mathbb{R}^d$,
- final constraints $c_f(t_f, y(t_f)) = 0$ with $c_f : \mathbb{R}^{d+1} \to \mathbb{R}^m$, $m \leq d$.
- possible state constraints $c(y(t)) \leq 0$.

The first source of uncertainty is the initial state y_0, which is unknown but within a set: $y_i \in \mathcal{Y}_i$. The second source of uncertainties is parameters.

2.1.2 Parameter uncertainty in the OCP.

Assuming that uncertainties are disturbances that depend on time gives a general framework as it allows the consideration of noises as well as estimation errors. However, this also makes the problem significantly more difficult in the nonlinear case: like in optimal control, the worst uncertainty might not be constant equal to a bound. This first creates an issue at a theory level as there are effectively two control inputs with two objectives. While this seems similar to minimax control [66] where $u(\cdot)$ minimizes the cost function and $\xi(\cdot)$ maximizes it, there is a significant difference in that trajectories that maximize the cost are not those that stray furthest from the nominal trajectory. Hence, computing $\xi(\cdot)$ with minimax control techniques will not yield an enclosure of all possible trajectories as will be highlighted by the example in Section 2.5. Instead, we consider that $\xi(\cdot)$ is chosen first, then the solution of the OCP is expressed as a function of $\xi(\cdot)$. As a consequence, we assume that the parameter function is Lipschitz so that the existence theorem and necessary optimality condition from Section 1.2 hold.

Once the solution of the OCP is stated for any $\xi(\cdot)$, we integrate every solution at once using set based methods.

However, this creates a second issue as our validated simulation library DynIbex [4] assumes that parameters are piecewise constant.

As a consequence, most of our experimental results are restricted to piecewise constant parameter function or to Problem 2.2 where parameters have unknown but constant values.

$$\begin{array}{l} \min\limits_{u(\cdot)} \displaystyle\int_0^{t_f} \ell(y(t),u(t),\xi)dt + \Psi(y(t_f)), \\ \text{s.t.} \left\{ \begin{array}{l} \dot{y}(t) = f(t,y(t),u(t),\xi), \\ y(i) \in \mathcal{Y}_i, \quad c_f(y(t_f)) = 0, \quad t_f \text{ is free.} \\ \quad \xi \in [\xi] \end{array} \right. \end{array} \tag{2.2}$$

Our first endeavor is enclosing the solutions of these OCPs, which is investigated in the next section.

2.2 Anticipative enclosure: bounding mathematically perfect solutions of the OCP

To have an accurate enclosure of the optimal control, we apply the PMP (see Section 1.2). Optimal trajectories and controls of Problem (2.1) are entirely characterized by Boundary Value Problem (BVP) (2.3).

$$\begin{cases} \dot{y}(t) = f(y(t), \arg\min_u H(y,p,u,\xi),\xi), \\ \dot{p}(t) = -\dfrac{\partial H}{\partial y}(y(t), \arg\min_u H(y,p,u,\xi),\xi), \\ y(t_i) \in \mathcal{Y}_i, \\ p(t_i) \text{ s.t } C_f(t_f, y(t_f), p(t_f)) = 0. \end{cases} \tag{2.3}$$

Geometrically, solving this OCP with uncertainties means taking the intersection of two sets. Indeed, for a time $\tau \in [t_i, t_f[$, if the state is such that $y(\tau) \in \mathcal{Y}_\tau$, with $\mathcal{Y}_\tau$ a subset of $\mathbb{R}^d$, then possible (state, co-state) couples are in the set $\mathcal{Y}_\tau \times \mathbb{R}^d$ which is the orange vertical strip on Figure 2.1. On the other hand, optimal (state, co-state) couples, which are solutions of (2.3), form another set, represented in blue. The solutions of the OCP lie in the intersection (in red) of these two sets.

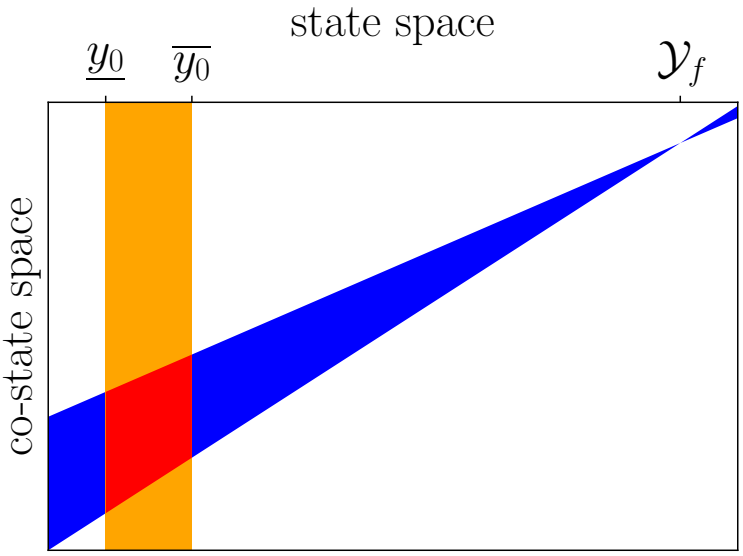


Figure 2.1: Geometric resolution of the OCP at time τ. These sets correspond to the solution of the example of Section 2.5.

For simplicity, we condense time, state and co-state of System (2.3) to obtain System (2.4).

$$\dot{x}(t) = g(x(t), \xi(t)),$$
$$x = \begin{pmatrix} t \\ y \\ p \end{pmatrix}, g = \begin{pmatrix} 1 \\ f(y, \arg\min H, \xi) \\ -\dfrac{\partial H}{\partial y}(y, p, \arg\min H, \xi). \end{pmatrix} \tag{2.4}$$

This system is hard to simulate because the term $\arg\min H$ creates non-continuous behaviors. We propose a hybrid formalism to account for that.

2.2.1 Hybrid formalism

Many controlled systems have hybrid dynamics that are either inherent to the system or emergent from the control minimization. Indeed, the dynamical system in (2.1) could be replaced by a hybrid controlled system as formulated in [33], to account for a multistage mission for instance. Even when the system is not hybrid, the optimality and state constraints induce hybrid behaviors such as bang-bang controls. For instance, the minimization of the Hamiltonian in Goddard's problem [19] yields the hybrid automaton of Figure 2.2.

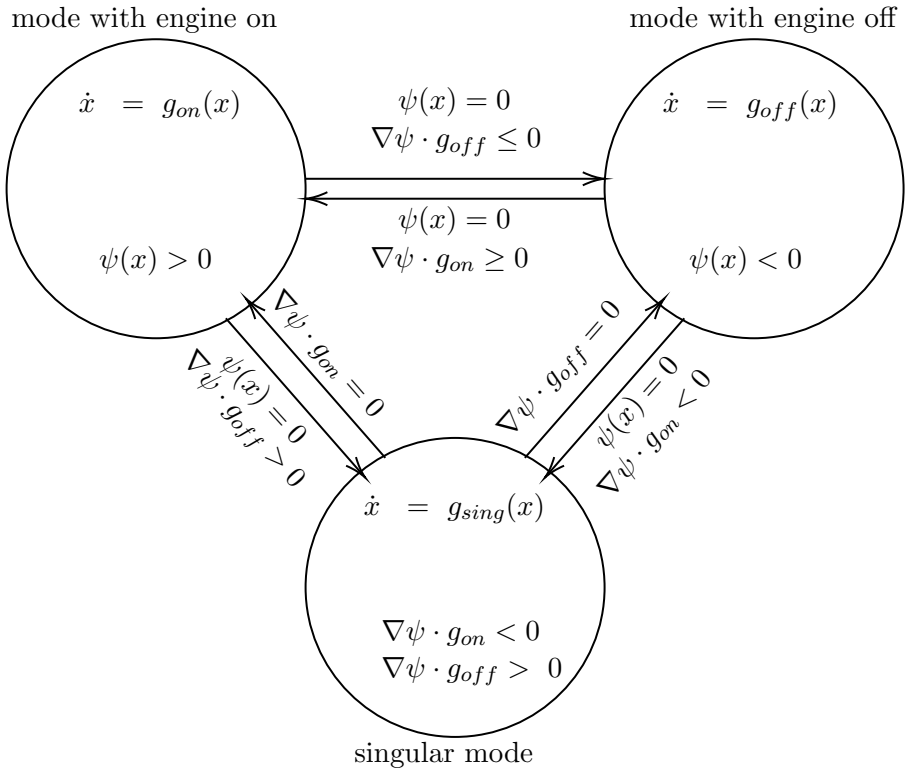


Figure 2.2: Hybrid structure emerging from Hamiltonian minimization for Goddard's problem [19]. There are no jumps, for each transition $x(t^+) = x(t^-)$.

Indeed, u minimizes $a(x) \cdot u + b(x)||u||$, meaning that the system alternates between three modes

- when $\psi(x) = ||a(x)|| - b(x) < 0$ then $u = 0$ which means the engine is off and the dynamic is g_{on},
- when $\psi(x) > 0$ then $||u|| = 1$ which means the engine is at maximum thrust and the dynamic is $g_{off} \neq g_{on}$,

- when $\psi(x) = 0$ and $0 \in [\nabla\psi \cdot g_{on}, \nabla\psi \cdot g_{on}]$ then u takes a value such that $\nabla\psi \cdot g_{sing} = 0$, this is a singular mode where the launcher cannot switch to one of the modes without immediately switching to the other (this avoids chattering).

However, this can be simplified into a single branch tree by using prior knowledge of the optimal trajectory structure, which may be achieved by analyzing the problem or solving it for nominal values of the parameters and initial state with numerical methods. For instance, Goddard's takeoff problem is known to start with engine on, then have a singular arc and finish with the engine off. Hence, this hybrid system can be simplified into the automaton on Figure 2.3.

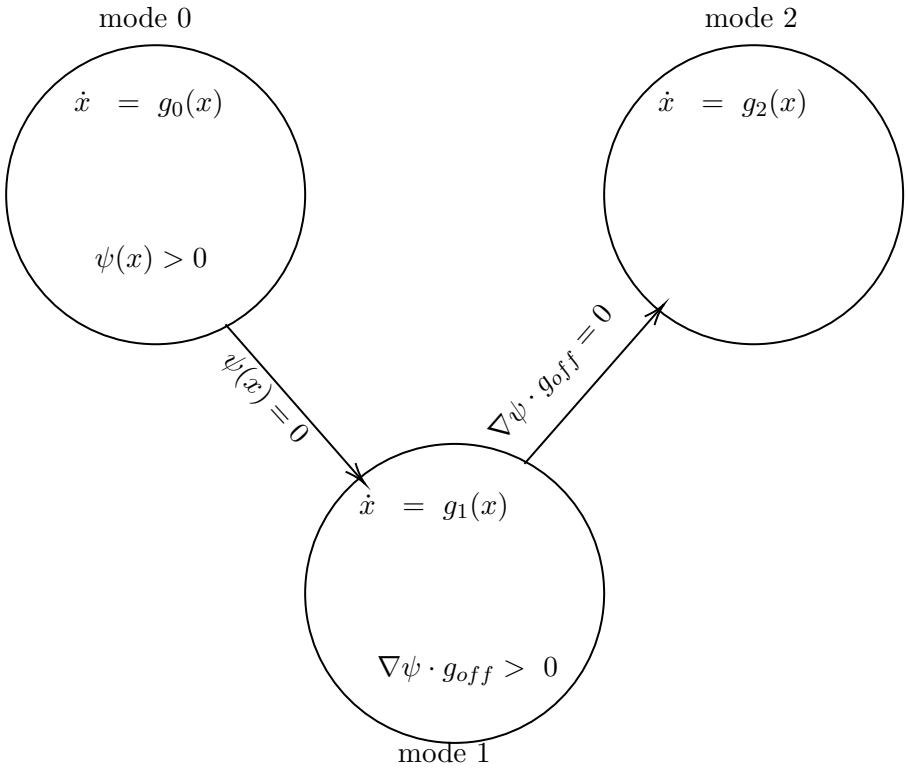


Figure 2.3: Simplification of Automaton 2.2 using knowledge of the optimal structure.

In this case the problem can be simplified by using a semi indirect method, *e.g.*, adding switch times as variables and switch conditions as boundary constraints. This results in the optimality condition of OCP (2.1) taking the form of a hybrid system with boundary constraints (2.5).

$$\begin{cases} \dot{x}(t) = g_n(x(t), \xi(t)), & \forall t \in [t_{n-1}^+, t_n^-] \\ x(t_n^+) = j_n(x(t_n^-)), & \forall n \in 1..N-1 \\ x(0) = x_0 \end{cases} \tag{2.5}$$

where $0 = t_0 < t_1 < ... < t_N = t_f$ are transition times and $j_n : \mathbb{R}^{1+2d} \mapsto \mathbb{R}^{1+2d}$ are jump functions. Switching and optimality conditions are compiled as Constraints (2.6).

$$C_n(x(t_n^-)) = 0, \forall n \in 1..N, \tag{2.6}$$

where $(C_n)_{n \in 1..N-1}$ are the previously mentioned switch functions, like $\psi(x) = 0$ on the first jump of the automaton given in figure 2.3, and $C_N(x(t_N^-)) = 0$ is the transversality condition $C_f(x(t_f)) = 0$. Uncertainties nudge the state which changes the time at which constraints are satisfied. Hence t_n vary depending on uncertainties.

To enclose the solutions of this problem, we usually use zero finding methods, which require to differentiate the BVP.

2.3 First derivative of the BVP

A BVP can be formalized as $\zeta(x_i, t_1...t_N) = 0$ using the flow of a hybrid system composed with constraint functions:

$$\zeta(x_i, t_1...t_f) = \begin{pmatrix} C_1(\Phi_{t_i,t_1}(x_i)) \\ C_2(\Phi_{t_i,t_2}(x_i)) \\ \vdots \\ C_N(\Phi_{t_i,t_N}(x_i)) \end{pmatrix}.$$

As noted in 1.1.3, the flow of a hybrid system $\Phi_{t_i,t_n}(x_i)$ can be written as a composition of flows and jump functions. For instance $\Phi_{t_i,t_2}(x_i) = \Phi_{t_1,t_2}(j_1(\Phi_{t_i,t_1}(x_i)))$. The derivative of a flow over any duration $[t_{n-1}^+, t_n^-]$ where the dynamics are differentiable is a resolvent matrix (see Section 1.1.3) and can be computed by simulating System (2.8). Hence, by chain rule, the first derivative of the BVP with respect to the initial state is:

$$\frac{\partial \zeta}{\partial x_i} = \begin{pmatrix} \nabla C_1(x_1) \cdot \mathbf{W}_1 \\ \nabla C_2(x_2) \cdot \mathbf{W}_2 \\ \vdots \\ \nabla C_N(x_N) \cdot \mathbf{W}_N \end{pmatrix}. \tag{2.7}$$

For every n, $\mathbf{W}_n = \mathbf{R}_{t_{n-1},t_n}(x_{n-1}) \cdot \nabla j_{n-1}(x_{n-1}) \cdot \mathbf{W}_{n-1}$, x_n^- and $\mathbf{R}_{t_{n-1},t_n}$ are the solution at time t_n of System (2.8) and $x_n^+ = j_n(x_n^-)$.

$$\begin{cases} \dot{x}(t) & = g_n(x,\xi) \\ \dot{\mathbf{R}}_{t_{n-1},t}(x_{n-1}) & = \dfrac{\partial f}{\partial x}(t,x(t)) \cdot \mathbf{R}_{t_{n-1},t}(x_{n-1}) \\ x(t_{n-1}) & = x_{n-1}^+ \\ \mathbf{R}_{t_{n-1},t_{n-1}}(x_n) & = I_d. \end{cases} \tag{2.8}$$

The derivative with respect to switch time can be deduced by the fact that delaying the switch time t_n means integrating dynamics g_n longer, hence, $\partial x(t_n^-)/\partial t_n = g_n(x_n^-)$. It also means integrating dynamics g_{n+1} less afterward so $\partial x(t_n^+)/\partial t_n = \nabla j_{n-1}(x_n^-) \cdot g_n(x_n^-) - g_{n+1}(x_n^+)$. It follows that:

$$\frac{\partial \zeta}{\partial t_i} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \nabla C_i(x_i) \cdot g_n(x_i^-) \\ \nabla C_{i+1}(x_{i+1}) \cdot \mathbf{W}_{i+1}^{t_i} \\ \vdots \\ \nabla C_N(x_N) \cdot \mathbf{W}_N^{t_i} \end{pmatrix},$$

with $\mathbf{W}_i^{t_i} = \nabla j_{i-1}(x_i^-) \cdot g_i(x_i^-) - g_{i+1}(x_i^+)$ and $\mathbf{W}_n^{t_i} = \mathbf{R}_{t_{n-1},t_n}(x_n) \cdot \nabla j_{n-1}(x_n) \cdot \mathbf{W}_{n-1}^{t_i}, \forall n > i$.

This will help to solve the BVP, thus compute trajectory sets.

2.3.1 Anticipative enclosure

We propose the following enclosure, which is an enclosure of optimal trajectories in the mathematical sense.

Definition 1. *For any parameter function $\xi(\cdot) : [t_i, t_f] \to [\xi]$ and any initial state $y_i \in \mathcal{Y}_i$, the anticipative enclosure contains the trajectory:*

$$\begin{cases} \dot{y}(t) = f(y(t), u(t), \xi(t)), \\ y(t_i) = y_i, \end{cases}$$

with $u(\cdot)$ the solution of the OCP:

$$\min_{u(\cdot)\in\mathcal{U}} \int_{t_i}^{t_f} \ell(y(t),u(t))dt + \Psi(y(t_f)) \ s.t. \ \begin{cases} \dot{y}(t) = f(y(t),u(t),\xi(t)) \\ y(t_i) = y_i. \end{cases},$$

To characterize the actual optimal trajectories, the following criterion is applied. If a trajectory is optimal from A to B, and C is an intermediate state of that trajectory, then the trajectory is optimal from A to C and from C to B. If A is the state at time τ, B is the state at time t_f and C is the state at time $\tau + dt$, then its (state, co-state) couple at time $\tau + dt$ satisfies the following two optimality properties.

- Optimality condition between A and C: the (state, co-state) couple lies in the right red polyhedron on Figure 2.4, which is also represented in purple polyhedron on Figure 2.5. This is the set obtained by integrating System (2.3) from time τ.

- Optimality condition between C and B: the (state, co-state) couple is in the blue cone on Figure 2.5, which is the set of solutions of BVP (2.3) with the initial time $\tau + dt$.

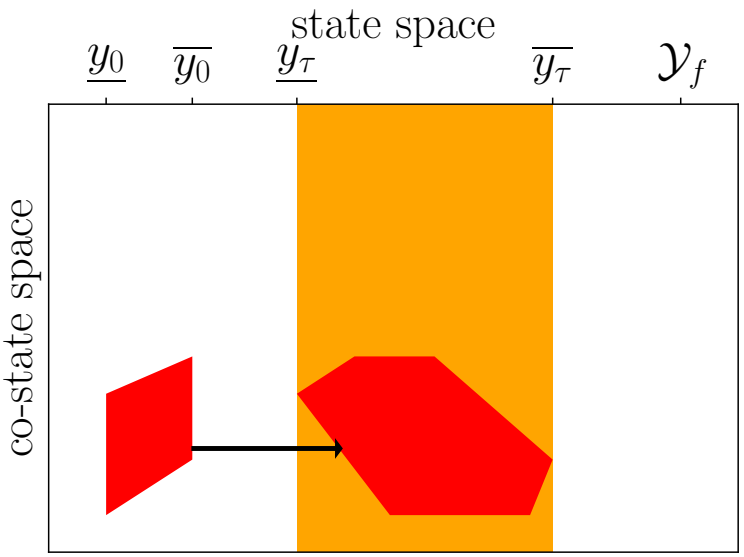


Figure 2.4: Simulation of the System (2.3) from time τ to time $\tau + dt$, starting with the red set obtained on Figure 2.1

By taking the intersection of these two sets, the set of optimal (state, co-state) couples is refined.

When doing this, the function $\xi(\cdot)$ corresponds to both the actual realization of the parameters and the estimation made by the system, so it is as if the vehicle has perfect knowledge of the uncertainties, even anticipating their future values. As such, this enclosure gives little information on the trajectory of a concrete system. In practice, parameter uncertainties will cause the system to deviate from its optimal trajectory and exit this enclosure. As a consequence, we propose the following two enclosures. They are meant to enclose the trajectory of a concrete system that computes its control by solving the OCP with an inaccurate model.

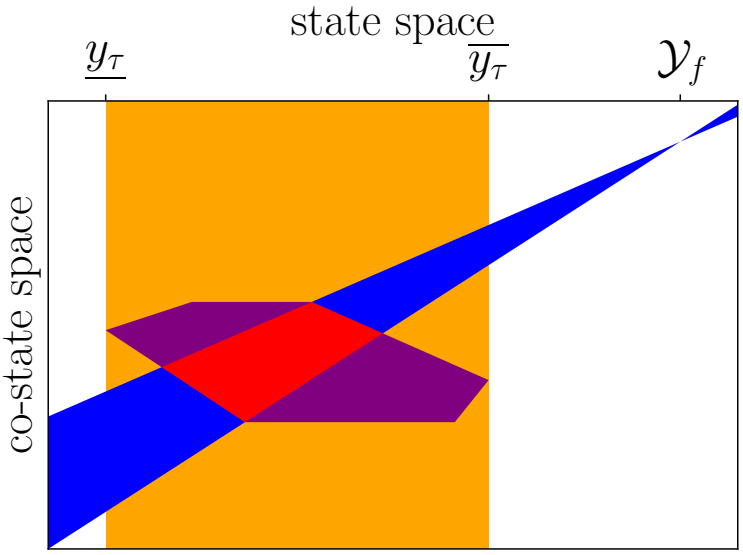


Figure 2.5: Anticipative: refining of the optimal couples at time $\tau + dt$ by intersecting the set of figure 2.4 with the blue set of optimal state, co-state couples.

2.4 Open-loop and closed-loop enclosures: bounding trajectories of a concrete system

In this section, we take into consideration that the controller might not have complete information in practice. We highlight two cases: a very pessimistic case where the system is blind and applies an incorrect control with no correction, and a less pessimistic case where the controller has information on the state and uses it to recompute the control. The latter has two variants: either the system recomputes its control at a few discrete instants, or it recomputes its control continuously.

2.4.1 Open-loop enclosure

In this section we consider the worst case scenario in which the controller computes a control with incorrect data and cannot correct afterward.

Definition 2. *The open-loop enclosure considers a system which blindly follows an initial solution that was computed with the nominal value of the parameters. For any two parameter functions $\xi(\cdot), \hat{\xi}(\cdot) : [t_i, t_f] \to [\xi]$, and any two initial states $y_i, \hat{y}_i \in \mathcal{Y}_i$, the open-loop enclosure contains the trajectory:*

$$\begin{cases} \dot{y}(t) = f(t, y(t), \hat{u}(t), \xi(t)) \\ y(t_i) = y_i, \end{cases}$$

with $\hat{u}(\cdot)$ the solution of the OCP:

$$\min_{\hat{u}(\cdot)\in\mathcal{U}} \int_0^T \ell(\hat{y}(t), \hat{u}(t))dt + \Psi(\hat{y}(T)) \ \ s.t. \ \begin{cases} \dot{\hat{y}}(t) = f(\hat{y}(t), \hat{u}(t), \hat{\xi}(t)) \\ \hat{y}(t_i) = y_i. \end{cases}$$

We assume that the system knows its initial state, but the actual realization of the parameters function $\xi(\cdot)$ is unknown. Hence, the system has an inaccurate estimation $\hat{\xi}(\cdot)$. This estimation can be the nominal value as is often the case in practice: $\forall t, \hat{\xi}(t) = \text{mid}([\xi])$. Alternatively, we can have $\hat{\xi}(\cdot) \in [\xi]$, which corresponds to a system that does parameter identification online. This second scenario is close to worst case analysis, since the worst case is applying a control tailored for an extreme value of the parameters to the opposite extreme value: the enclosure will contain the worst under or over-shooting scenarios. This will however create interesting properties, notably that the anticipative enclosure is contained in the open-loop enclosure.

The control is deduced by the state and co-state of the PMP. However, the state in this equation does not correspond to the actual state, rather to an estimation.

This can be formulated as the following system:

$$\begin{cases} \dot{y}(t) = f(t, y(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \xi(\cdot)) \\ \dot{\hat{y}}(t) = f(t, \hat{y}(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \hat{\xi}(t)) \\ \dot{\hat{p}}(t) = -\dfrac{\partial H}{\partial y}(t, \hat{y}(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \hat{\xi}(t)) \\ y(t_i) = y_i \in \mathcal{Y}_i, \\ \hat{y}(t_i) = y_i, \\ \hat{p}(t_i) \text{ solution of IVP } C(\hat{t}_f, \hat{y}(t_f), \hat{p}(t_f)) = 0. \end{cases} \tag{2.9}$$

Hence, this open-loop enclosure can be built by first computing the anticipative enclosure, then integrating trajectories alongside it. This yield as very pessimistic enclosure as can be seen with our example system in Figure 2.7.

A less pessimistic enclosure is built by taking into account the fact that the system can correct its trajectory online using sensor data.

2.4.2 Closed-loop enclosure with finite recomputations

Some systems recompute their trajectory online. For instance reusable launchers solve an OCP right before the landing burn to compute a new guidance trajectory tailored for their actual position [16]. We propose the following enclosure to take that into account.

Definition 3. *The discrete closed-loop enclosure considers a system that uses a perfect measure of its state to recompute the solution of the OCP online (but with inaccurate parameters). Consider a list of recomputation time* $(\tau_k)_{k\in 0..K}$, $t_i = \tau_0 < \tau_1 .. < \tau_K = t_f$. *For any two parameter functions* $\xi(\cdot), \hat{\xi}(\cdot) : [t_i, t_f] \to [\xi]$, *and any initial states* $y_i \in \mathcal{Y}_i$, *the closed-loop enclosure contains the trajectory of the piecewise-defined system:*

$$\begin{cases} \dot{y}(t) = f(t, y(t), \hat{u}_k(t), \xi(t)), \forall t \in [\tau_k, \tau_{k+1}] \\ y(t_i) = y_i, \end{cases}$$

with $\hat{u}_k(\cdot)$ *the solution of the OCP:*

$$\min_{\hat{u}(\cdot)\in\mathcal{U}} \int_{\tau_k}^{T} \ell(\hat{y}(t), \hat{u}(t))dt + \Psi(\hat{y}(T)) \text{ s.t. } \begin{cases} \dot{\hat{y}}(t) = f(t, \hat{y}(t), \hat{u}(t), \hat{\xi}(t)) \\ \hat{y}(\tau_k) = y(\tau_k). \end{cases}$$

The overall control is made of pieces $\hat{u}_k(\cdot)$ that are computed with an accurate measurement of the state $y(\tau_k)$. As in the open-loop enclosure, $\xi(\cdot)$ corresponds to the actual realization of the parameters, which is unknown, and $\hat{\xi}(\cdot)$ is its inaccurate estimation. The goal is to enclose the actual operation of a system with an optimal control regulator. The system does not have access to the value of the parameters, but it compensates using measures of its state.

Recomputations can be simulated by resetting the estimated state and optimal co-state at each recomputation time, yielding:

$$\begin{cases} \dot{y}(t) = f(t, y(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \xi(\cdot)) \\ \dot{\hat{y}}(t) = f(t, \hat{y}(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \hat{\xi}(t)) \\ \dot{\hat{p}}(t) = -\dfrac{\partial H}{\partial y}(t, \hat{y}(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \hat{\xi}(t)) \\ y(i) \in \mathcal{Y}_i, \\ \forall t_k, \hat{y}(t_k) = y(t_k), \\ \forall t_k, \hat{p}(t_k) \text{ solution of BVP } C(\hat{t}_f, \hat{y}(t_f), \hat{p}(t_f)) = 0. \end{cases} \tag{2.10}$$

Note that the equation $\hat{y}(t_k) = y(t_k)$ assumes a perfect observer and a more realistic system can be built by replacing this equality by a concrete observer. Note also that in between two recomputation times, Dynamics (2.10) are the same as open-loop Dynamics (2.10). This is due to the fact that when no sensor data is available, the system is effectively in open-loop.

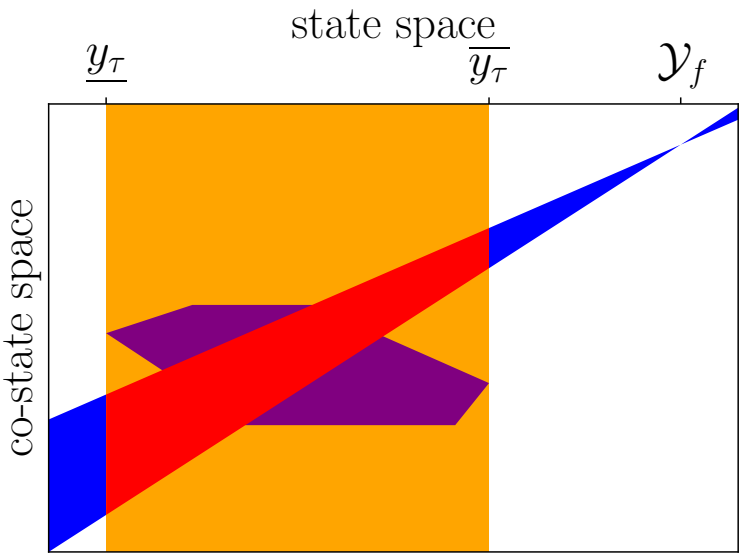


Figure 2.6: Closed-loop: recomputation of the optimal co-states at time $\tau + dt$.

If the system recomputes its control at $\tau + dt$ using accurate measurements from the sensors, the OCP is solved again with an initial state in $[y_{\tau+dt}]$. This is illustrated in Figure 2.6, which is similar to Figure 2.4. An important point is that the red set of recomputed (state, co-state) couples is not contained in the purple set obtained by integrating System (2.3) from a prior step. This is due to the fact that the prior co-states were computed with inaccurate parameters. Replacing them is a correction. Hence, these trajectories do not respect the equations of the PMP: applying a control that is optimal for a faulty model results in a non optimal trajectory.

This is exacerbated when the frequency of the controller is great compared to the duration of the mission, in which case the system can be better represented by continuous recomputation times.

2.4.3 Closed-loop enclosure with continuous recomputation

When the frequency of recomputation time is high, recomputation itself can be approximated by a differential algebraic equation.

$$\begin{cases} \dot{y}(t) = f(t, y(t), \arg\min_u H(\hat{y}, \hat{p}, u, \hat{\xi}(t)), \xi(\cdot)) \\ y(t_i) \in \mathcal{Y}_i, \\ \forall t, \hat{y}(t) = y(t), \\ \forall t, \hat{p}(t) \text{ solution of BVP } C(\hat{t}_f, \hat{y}(t_f), \hat{p}(t_f)) = 0. \end{cases}$$

The co-state has no derivative as it is defined implicitly at all time.

To have a general idea of what sets those definitions correspond to, we showcase them on a controlled integrator with quadratic cost.

2.5 Application to an integrator

In this section we illustrate the three enclosures on a simple problem. Consider a simple integrator with quadratic cost:

$$\min_{u(\cdot)\in\mathcal{U}} \int_\tau^{t_f} \frac{u^2}{2} dt \text{ s.t. } \begin{cases} \dot{y}(t) = \xi(t)u, \\ y(t_i) = y_i, \\ y(t_f) = y_f, \\ t_f \text{ is fixed.} \end{cases}$$

We use this system because it can easily be displayed on a graph, and an analytical solution can be computed.

Indeed, the application of PMP yields:

$$\begin{aligned} &H(y,p,u) = \tfrac{u^2}{2} + p\xi u, \\ &\dot{p}(t) = -\tfrac{\partial H}{\partial y} = 0, \\ &u = \min_u H = -p\xi. \end{aligned}$$

Combining these equations yields BVP:

$$\begin{cases} \dot{y}(t) = -\xi(t)^2 p, \\ \dot{p}(t) = 0, \\ y(t_f) = y_f, p(t_f) \text{ is free.} \end{cases}$$

The analytical solutions of the optimal state and co-state are:

$$\begin{aligned} p(t) &= -\tfrac{y_f - y(\tau)}{\int_\tau^{t_f} \xi^2} && \in -\tfrac{y_f - y(\tau)}{t_f - \tau}[\tfrac{1}{\overline{\xi}^2}, \tfrac{1}{\underline{\xi}^2}], \\ y(t) &= y(t_i) + \tfrac{\int_{t_i}^t \xi^2}{\int_0^{t_f} \xi^2}(y_f - y(t_i)) && \in y(t_i) + t\tfrac{y_f - y(t_i)}{t_f - t_i}[\tfrac{\underline{\xi}^2}{\overline{\xi}^2}, \tfrac{\overline{\xi}^2}{\underline{\xi}^2}], \end{aligned} \tag{2.11}$$

which we use to draw the anticipative enclosure.

Note that if the parameter is constant, then anticipative trajectories simplify as a straight line $y(t) = (t_f - t)/(t_f - t_i)y_i - y_f$, which means that the anticipative enclosure is a cone converging to the target no matter the uncertainties on ξ. However, when the parameter is variable, the anticipative enclosure is oval, as can be seen on Figure 2.7. Moreover, the boundary of the enclosure does not correspond to two extreme trajectories, rather each point is the extreme point at time τ of the trajectory corresponding to parameter $\xi(t) = \underline{\xi}, \forall t < \tau$ and $\xi(t) = \overline{\xi}, \forall t > \tau$. This is due to a rebound effect, when parameters change, the control adapts, here it lies behind when the value of the parameter is low and advances faster when the parameter is high.

Note also that the worst case parameter in the minimax sense is $\xi(t) = \underline{\xi}, \forall t$, as it will achieve the worst performance on the cost. But is not an extreme deviation: it will yield a straight trajectory from y_i to y_f. This illustrates the fact that final cost and deviation are two different criteria that are generally not correlated.

The open-loop enclosure is expressed by taking the set of possible initial co-states using Equation 2.11 at time t_i, then integrating forward without ever correcting the co-state. This lead to a cone shape of Figure 2.7 that misses its target.

In continuous closed-loop, Equation 2.11 yields:

$$\begin{aligned} \dot{y}_{max} &= \frac{\overline{\xi}^2}{(t_f - t)\underline{\xi}^2}(y_f - y_{max}) \\ \dot{y}_{min} &= \frac{\underline{\xi}^2}{(t_f - t)\overline{\xi}^2}(y_f - y_{min}) \end{aligned} \tag{2.12}$$

Indeed, both can be rewritten as $\dot{y} = \frac{K}{t_f - t}(y_f - y)$. If we take $z = (T - t)^{-K}(y - y_f)$ then $\dot{z} = 0$. Hence, $(T - t)^{-K}(y(t) - y_f) = T^{-K}(y(t_i) - y_f)$ et $y(t) = y_f + (\frac{t_f - t}{T})^K(y(t_i) - y_f))$.

This leads to a bulge-shaped enclosure in Figure 2.7 with a vertical tangent at the final time, which correspond to the dangerous scenario where a system spends a lot of energy to correct all accumulated error within a very short time frame.

All enclosures in Figure 2.7 are computed with the following uncertainties: $y_i \in [-0.5, 0.5]$ and $\xi \in [9, 11]$.

Most dynamical systems do not have analytical solutions. Hence, the next two sections focus on the design of set-based algorithms to compute these three enclosures.

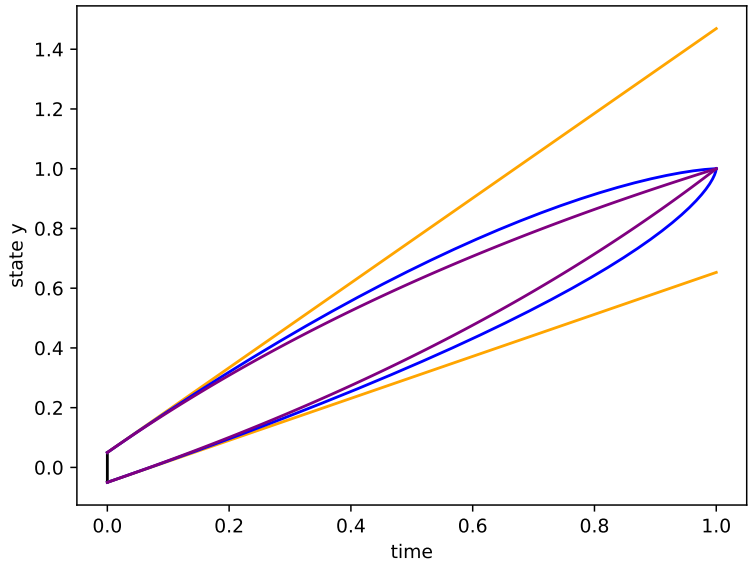


Figure 2.7: Enclosures of optimal trajectories for an integrator with quadratic cost with uncertainties. The anticipative enclosure delimited by the purple curves converges to the target. The continuous closed-loop enclosure delimited by the blue curves also converges to the target, but it deviates at the beginning due to flawed estimation and corrects its trajectory abruptly at the end, leading to bulge shape and a very steep tangent at the final time. The open-loop enclosure delimited by the orange curves completely overshoots and undershoots its target. Those are the patterns we seek to evaluate with our enclosures: how hard will the system correct its trajectory if it works in closed-loop? How far will it land from its target if it works in open-loop?

Chapter 3

Development of interval-based methods to enclose sets of optimal trajectories

This chapter presents our first method to compute the three enclosures defined in Chapter 2. This method is the main contribution of our first publication [13], it uses intervals exclusively, hence Optimal Control Problem (OCP) (2.1) is changed so that all uncertainties are intervals. We also restrict ourselves to the case with no variable durations for reasons that will be discussed in Section 3.3.1, which means the final time is fixed, and the system is not hybrid. This yields OCP (3.1):

$$\begin{array}{l} \min\limits_{u(\cdot)} \displaystyle\int_{t_i}^{t_f} \ell(y(t),u(t),\xi(t))dt + \Psi(y(t_f)), \\ \text{s.t.} \left\{ \begin{array}{l} \dot{y}(t) = f(y(t),u(t),\xi(t)), \\ y(t_i) \in [y_i], \quad c_f(t_f,y(t_f)) = 0, \quad t_f \text{ is fixed}, \\ \xi(t) \in [\xi], \end{array} \right. \end{array} \tag{3.1}$$

where f, ℓ and $\mathcal{U}$ are such that no hybrid dynamics emerge from the control minimization.

To enclose optimal trajectories, we propose a contractor-based OCP solver that encloses the set of possible initial co-states around a nominal solution. Then this solver is embedded in three algorithms, one for each enclosure of Chapter 2. This however runs into difficulties due to box sets retaining no correlation between each possible state and their associated co-state, which we alleviate with paving. This method is applied to a low dimension system, and we discuss its shortcomings, which set the objectives for our later work presented in Chapter 4.

3.1 Interval-based OCP solver

The first step to enclose optimal trajectories is to compute the set of optimal (state, co-state) couples, that is the red set on Figure 3.1 (which is a recall of Figure 2.1).

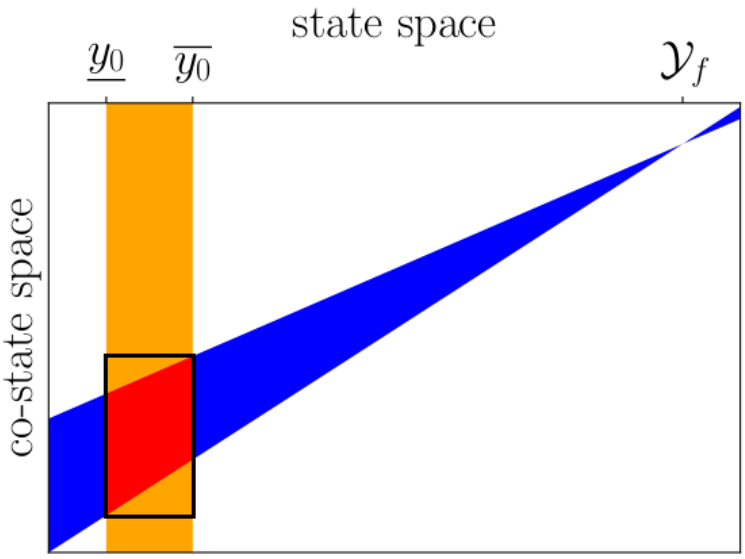


Figure 3.1: The set of possible optimal state co-state couples is enclosed in a box. The initial state box (horizontal axis) is given so the unknown is the co-state box (vertical axis).

To that end, we design a contractor-based method. First, we formulate Krawczyk's contractor for the optimality condition derived from Pontryagin's Minimum Principle (PMP) which is recalled in System (3.2) and use validated simulation to compute it. This contractor is then used in a classic Inflate&Contract algorithm to compute all possible optimal (state, co-state) couples around nominal solutions.

$$\begin{cases} \dot{y}(t) = f(y(t), \arg\min_u H(y,p,u,\xi), \xi), \\ \dot{p}(t) = -\dfrac{\partial H}{\partial y}(y(t), \arg\min_u H(y,p,u,\xi), \xi), \\ y(t_i) \in [y_i], \\ p(t_i) \text{ s.t } C(t_f, y(t_f), p(t_f)) = 0. \end{cases} \tag{3.2}$$

3.1.1 A Krawczyk contractor for OCPs

The aim is to enclose all possible optimal trajectories spanning from a given state at time $\tau \in [t_i, t_f[$: either the initial time or an intermediary recomputation time (the final time t_f is excluded).

Optimal trajectories verify the optimality condition given by PMP. Because we restrict ourselves to the case with fixed final time and no hybrid behaviors, there is no intermediate switching time, hence condition (2.6) consists of a single boundary condition which can be written as:

$$\zeta(p) = C(\Phi_{\tau,t_f}(\tau, y, p)) = 0,$$

where $\Phi_{\tau,t_f}(\tau, y, p)$ is the flow of System (3.2). Hence, for a given enclosure of the state $[y_\tau]$, a Krawczyk operator of the function $y, p \to C(\tau, y_\tau, p)$ is built using Krawczyk Formula (1.13) and the first derivative of the BVP proposed in Section 2.3.

$$\mathcal{C}_K([p_\tau]) = [p_\tau] \cap \left(\begin{array}{c} \text{mid}\,([p_\tau]) + M \cdot [\zeta](\text{mid}\,([p_\tau]))+ \\ \left(I_d - M \cdot \left[\frac{\partial \zeta}{\partial p_\tau}\right]([p_\tau])\right) \cdot ([p_\tau] - \text{mid}\,([p_\tau])) \end{array} \right), \quad (3.3)$$

where:

- $[\zeta](\text{mid}\,([p_\tau])) = [C]\left(\left[\Phi_{\tau,t_f}\right](\tau, [y_\tau], \text{mid}\,([p_\tau]))\right)$,
- $\left[\frac{\partial \zeta}{\partial p_\tau}\right]([p_\tau]) = [\nabla C] \cdot [\mathbf{R}]^p_{\tau,t_f}$,
- $M = \text{mid}\left(\left[\frac{\partial \zeta}{\partial p_\tau}\right]([p_\tau])\right)^{-1}$.

Unlike in Section 2.3, we only differentiate to the initial co-state, hence we only enclose $\mathbf{R}^p_{\tau,t} = \partial \Phi_{\tau,t}/\partial p$. This is computed by integrating ODE (3.4), which is an interval counterpart of ODE (2.8).

$$\left\{ \begin{array}{ll} \dot{x}(t) & \in [g](x, [\xi]), \\ \dot{\mathbf{R}}^p_{\tau,t} & \in \left[\frac{\partial g}{\partial x}\right](x, [\xi]) \cdot \mathbf{R}^p_{\tau,t} \\ x(\tau) & \in (0; [y_\tau]; [p_\tau]), \\ \mathbf{R}^p_{\tau,\tau} & = (0; I_d). \end{array} \right. \quad (3.4)$$

The enclosure of the midpoint flow $[\Phi_{\tau,t_f}]([y_\tau], \text{mid}\,([p_\tau]))$ can be computed by simulating:

$$\left\{ \begin{array}{ll} \dot{x}(t) & \in [g](x, [\xi]), \\ x(\tau) & \in (0; [y_\tau]; \text{mid}\,([p_\tau])), \end{array} \right. \quad (3.5)$$

This results in Algorithm 1 that computes Krawczyk's contractor for given states and co states.

Algorithm 1: Computes $\mathcal{C}_K([p_\tau])$ for given $[y_\tau]$ and $[p_\tau]$.

Data: $[y_\tau]$ and $[p_\tau]$.
Result: $\mathcal{C}_K([p_\tau])$.
1 Compute $[\mathbf{R}]^p_{\tau,t_f}([y_\tau],[p_\tau])$ by simulating (3.4).
2 Compute $\left[\Phi_{\tau,t_f}\right](\tau,[y_\tau],\mathrm{mid}([p_\tau]))$ by simulating (3.5).
3 Compute $\mathcal{C}_K([p_\tau])$ using Formula (3.3).

This algorithm requires two forward simulations with fixed horizons, hence it will terminate because validated simulation always terminates, even if that entails outputting useless sets. Indeed, in the case where validated simulation cannot simulate the system, because of an explosion of the state set notably, it terminates regardless and outputs the entire space: $[y(t_f), p(t_f)] \in \mathbb{R}^{2d}$. The resulting set $\mathcal{C}_K([p_\tau])$ is guaranteed to enclose all possible solutions in the inputted initial box as a consequence of contractor properties (see Section 1.3.1 or [40]). However, it will miss solutions that are outside the initial box. As a consequence this contractor is embedded in an iterative algorithm.

3.1.2 Inflate&contract method around nominal solution

Because Krawczyk contractor is such that $\mathcal{C}_K([p_\tau])$ encloses all optimal co-states in $[p_\tau]$, we have the following properties:

- $\mathcal{C}_K([p_\tau]) \subseteq [p_\tau]$,
- $\mathcal{C}_K([p_\tau])$ contains all optimal co-states in $[p_\tau]$,
- If $\mathcal{C}_K([p_\tau]) \subset Int([p_\tau])$, the interior of $[p_\tau]$, then $\mathcal{C}_K([p_\tau])$ contains all optimal co-states, or at least all connected local optimums.

Indeed, the set of optimal co-states are locally connected because changing parameters and initial state changes the OCP continuously, and changing an OCP continuously changes its solution continuously [5]. As noted in Section 1.3.1, if there are solutions outside the box, then the solution set touches the border, hence it is not in its interior, so if $\mathcal{C}_K([p_\tau]) \subset Int([p_\tau])$ then it is guaranteed to enclose all local solutions. However, there might be multiple separate solutions sets. A typical example is planning a car trajectory around an obstacle in the middle of the road. There will be one set of co-states that corresponds to the car passing the obstacle on the right, and the other set of co-states corresponds to the co-state passing the obstacle on the left. We restrict ourselves to connected local optimums as the first order optimality condition given by PMP is only a local condition, hence we have little means to find solutions disconnected from the nominal solution. Moreover, even if we were given a set of disconnected solutions, enclosing all of them with a single box would probably be counter-productive.

In the car example, enclosing both sets of co-states in a single box would yield trajectories that go head first into the obstacle. Better results would be achieved by considering connected local optimums around each of these solutions individually.

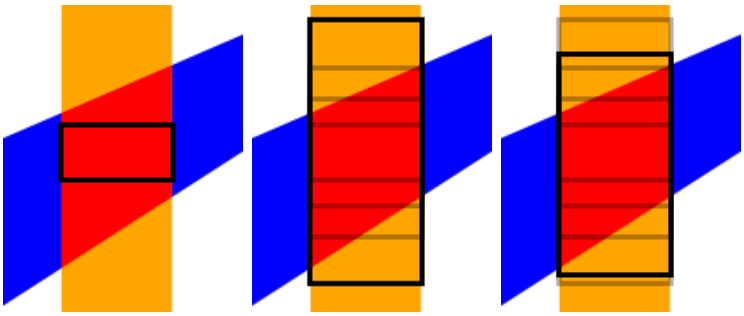


Figure 3.2: Inflate&Contract method. i) Take an initial box $[p_\tau]$ around solutions for nominal values of the uncertainties. ii) Inflate it until it encloses the entire solution set, which is checked with sufficient condition $\mathcal{C}_K([p_\tau]) \subset Int([p_\tau])$. iii) Contract with fixed point iteration $[p_\tau] \leftarrow \mathcal{C}_K([p_\tau])$. Note that this procedure is only applied along the vertical axis, which corresponds to co-state space because the state is an input.

To compute those connected local optimums, Algorithm 2 is used. It is a classic inflate and contract algorithm specialized to enclose initial co-states and is illustrated on Figure 3.2.

Algorithm 2: Computes a thin enclosure $[p_\tau]$ of the optimal co-states corresponding to a given state enclosure $[y_\tau]$.

Data: $[y_\tau]$ and an initial searching area $[p_\tau]$.
Result: $[p_\tau]$.
1 Compute $\mathcal{C}_K([p_\tau])$ using algorithm 1.
2 **while** $\mathcal{C}_K([p_\tau]) \not\subset Int([p_\tau])$ **do**
3 Inflate $[p_\tau]$ by a coefficient λ.
4 Compute $\mathcal{C}_K([p_\tau])$ using algorithm 1.
5 **while** $D_{Hausdorff}([p_\tau], \mathcal{C}_K([p_\tau])) > precision$ **do**
6 $[p_\tau] \leftarrow \mathcal{C}_K([p_\tau])$.
7 Compute $\mathcal{C}_K([p_\tau])$ using algorithm 1.

Inflating an interval is the operation that increases it in both direction: $[\underline{a}, \overline{a}] \rightarrow [\underline{a} - h, \overline{a} + h]$, with $h \in \mathbb{R}^+$. We typically take h proportional to the radius of the interval $h = \lambda(\overline{a} - \underline{a})$. Inflating a box on Line 3 means inflating each of its interval coordinates. The convergence criterion is Hausdorff distance, which measures to how much one box needs to be inflated to fit the other box and vice versa.

Algorithm 2 is not guaranteed to converge: the convergence criterion on Line 2 might never be met, and the box can be inflated endlessly. This is due to the fact that over-approximation increases with the size of the search box and past a certain threshold it overtakes the contraction of optimality conditions. This phenomenon is key in the determination of inflate coefficient λ on Line 3: if λ is big, the inflating part converges faster, unless it is too big in which case the inflating part does not converge at all. As a consequence it is necessary to add an iteration counter and cap it to ensure that the program terminates, with the program returning $\mathbb{R}^d$ if the maximum iteration is reached. This convergence issue is one of the main weakness of our method and discussed in depth in Chapter 5. Nevertheless, when the inflating part does converge, the contracting part is guaranteed to converge as well (see [40]) and the result encloses all connected solutions, as explained in previous paragraphs.

Unlike in the real case, an enclosure of optimal states and co-states at a given time is not enough to have an enclosure at all subsequent times. This will be discussed in next section where Algorithm 2 is paired with validated simulation to compute the three enclosures.

3.2 Computation of the enclosures

In this section, interval-based methods are designed to enclose each enclosure proposed in Chapter 2. We first highlight that naive implementations encounter a significant wrapping effect due to a loss of correlation between the solver and simulation. We then propose paving-based algorithms for open-loop, closed-loop and anticipative enclosures.

3.2.1 The necessity of paving

The naive implementation to compute enclosures consists in first enclosing optimal co-states with Algorithm 2 then integrating PMP dynamics with validated simulation and solving the OCP again when in closed loop.

However, validated simulation suffers from wrapping effect due to a loss of correlation between optimal state and co-state as illustrated on Figure 3.3. Indeed, it implicitly applies a co-state (hence a control) tailored for extreme state to the other extreme state, leading to significant over-approximation.

This is especially troublesome for closed-loop algorithm as recomputing the co-state to simulate the controller correcting the trajectory actually increases the size of the enclosure. This phenomenon is illustrated on Figure 3.4 where the box obtained by recomputing the co-state at an intermediate time does not filter out open-loop trajectories.

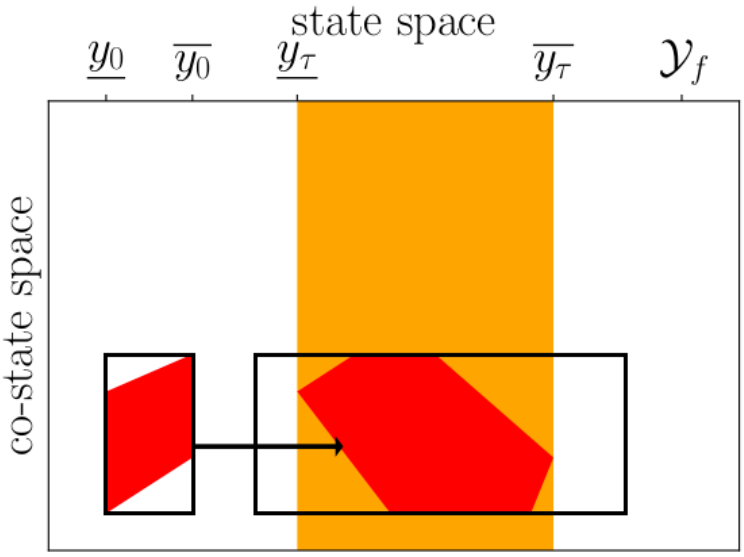


Figure 3.3: Callback to Figure 2.4. The right black box is obtained with validated simulation and significantly overestimates the orange stripe of possible states.

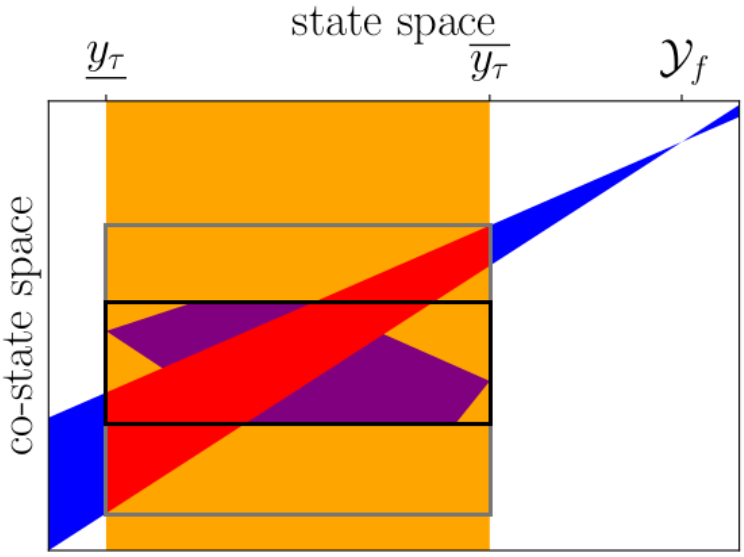


Figure 3.4: Purple: (state, co-state) couples obtained by simulating solutions from previous time. Black: box enclosing the purple set obtained by integrating the system (left of Figure 3.3). Red: optimal couples. Grey: results of our OCP solver. Parts of the purple set that are outside of the blue set are no longer optimal and should be culled. Because the gray box does not exclude this purple set and is larger than the black box, it will be more pessimistic than open-loop.

To alleviate this wrapping effect, paving is used. Paving consists in splitting the box into a set of smaller boxes. The gain of precision is showcased on Figures 3.5 and 3.6. This reduces wrapping effect enough to compute our closed-loop enclosure.

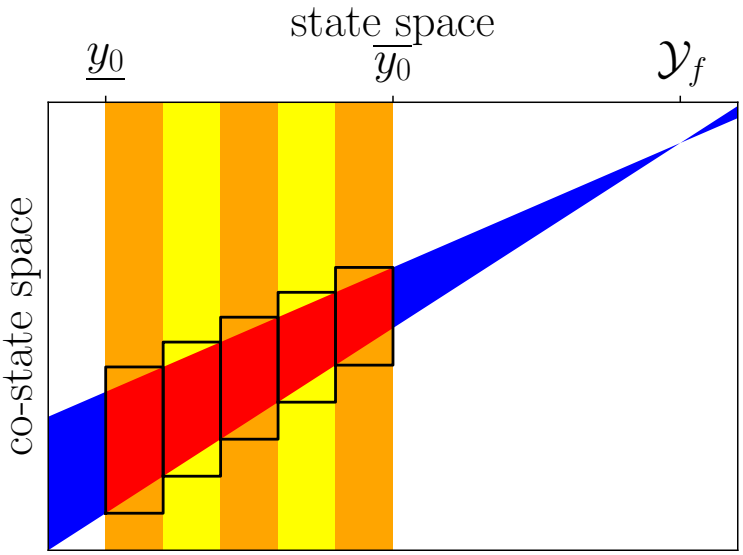


Figure 3.5: Paving the state enclosure to get a thinner approximation.

However, paving comes at the cost of significant computation time in high dimension. Indeed, splitting a box of dimension d in l parts in each direction yields l^d boxes.

Nevertheless, this allows the computation of enclosures in low dimension which is presented in the next section.

3.2.2 Paving-based algorithms to compute the three enclosures in low dimension

In this section, we present the first three algorithms we developed to compute the three enclosures defined in Chapter 2.

We first tackle the open-loop enclosure as it is the simplest. Indeed, it is sufficient to enclose the solution of the OCP with Algorithm 2 then simulate open-loop Dynamics 2.9. To achieve better performance this is done not on the entire box $[y_i]$ but rather on a paving $([y_i]_k)_{k\in 0..K}$ (see previous section). This yields Algorithm 3 which uses temporary variables $[y]$, $[p]$ that are marked without indexes. The paving $([y_i]_k)_{k\in 0..K}$ is obtained by splitting each interval coordinate of $[y_i]$ in l sub intervals of equal length, yielding $K = l^d$ boxes, then organizing them in a single sequence for ease of use. However, we do not keep every result of every box of the paving in memory as it could potentially take a lot of space and be hard to visualize.

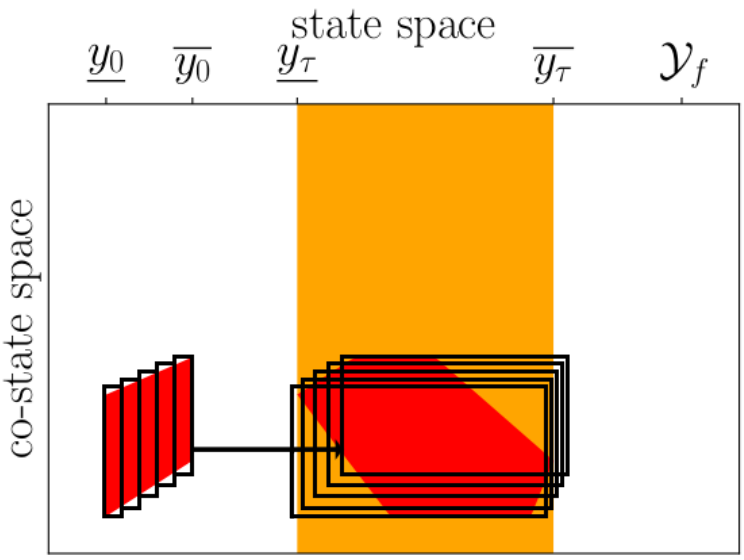


Figure 3.6: The OCP is solved and simulated for one part at a time. The union of the boxes is considerably tighter than the box on Figure 3.3.

Rather we compute the union of every result incrementally (see lines 7, 14, 15, 16 and 17). Note that despite the wrapping effect this union creates, paving still yields a tighter enclosure as can be seen on Figure 3.6.

Algorithm 3 is composed of calls to Algorithm 2 followed by validated simulation on a fixed duration. As such it terminates as long as Algorithm 2 does. Its output, the sequence of boxes $([y_n])_{n\in 0..N}$, and Picard boxes $([\tilde{y}_n])_{n\in 0..N-1}$, is guaranteed to enclose open-loop trajectories because Algorithm 2 encloses all optimal initial co-states, which includes the one that will be used by the system's controller, then validated simulation encloses all trajectory of open-loop Dynamics (2.9). It is recalled that open-loop Dynamics (2.9) have two states, the true state $[y]$ of the system and its estimation $[\hat{y}]$, and the system computes its control with the estimation $[\hat{y}]$ and its associated co-state $[\hat{p}]$, rather than the true state, because it does not know it (see Chapter 2 for details). Algorithm 3 does not output the enclosure of the estimated state and co-state because those are mathematically perfect trajectories, hence can be computed more accurately with Algorithm 6 that is dedicated to anticipative enclosure.

To compute the closed-loop enclosure, the problem needs to be recomputed at times of interest. Since each recomputation requires a dedicated paving, the paving loop on variable k and the time loop on n are swapped. This yields Algorithm 4. Like Algorithm 3, it terminates as long as Algorithm 2 does.

Algorithm 3: Paving-based algorithm for the open-loop enclosure.

Data: $[y_i]$, times of interest $t_i < \tau_1 < ... < \tau_{N-1} < t_f$.
Result: $([y_n])_{n\in 0..N}$ enclosures at times τ_n, $([\tilde{y}_n])_{n\in 0..N-1}$ enclosures over $[\tau_n, \tau_{n+1}]$.

$([y_n])_{n\in 0..N}, ([p_n])_{n\in 0..N}, ([\tilde{y}_n])_{n\in 0..N-1}, ([\tilde{p}_n])_{n\in 0..N-1} \leftarrow \emptyset$.
$[y_0] \leftarrow [y_i]$.
Split $[y_i]$ in a paving $([y_i]_k)_{k\in 0..K-1}$.
$k \leftarrow 0$.
while $k < K$ **do**
  $[\hat{p}] \leftarrow$ Algorithm 2 with $[y_i]_k$.
  $[\hat{p}_0] \leftarrow [\hat{p}_0] \cup [\hat{p}]$.
  $[y] \leftarrow [y_i]_k$.
  $[\hat{y}] \leftarrow [y_i]_k$.
  $n \leftarrow 0$.
  while $n < N$ **do**
    Simulate System (2.9) from τ_n to τ_{n+1} with initial state $([y], [\hat{y}], [\hat{p}])$.
    $([y], [\hat{y}], [\hat{p}]) \leftarrow$ end state of simulation.
    $[y_{n+1}] \leftarrow [y_{n+1}] \cup [y]$.
    $[\hat{y}_{n+1}] \leftarrow [\hat{y}_{n+1}] \cup [\hat{y}]$.
    $[\hat{p}_{n+1}] \leftarrow [\hat{p}_{n+1}] \cup [\hat{p}]$.
    $([\tilde{y}_n], [\tilde{p}_n]) \leftarrow ([\tilde{y}_n], [\tilde{p}_n]) \cup$ Picard boxes of simulation.
    $n \leftarrow n + 1$.
  end
  $k \leftarrow k + 1$.
end

However, Algorithm 2 is much more likely to not converge at some point. Indeed, co-states are enclosed not only at the initial time but also later on after the system deviated from its nominal course due to uncertainties. As a consequence it is possible to reach the limit of the admissible set, leading to abnormal trajectories, leading to unbounded co-states and causing Algorithm 2 to not converge. This is a recurring issue in tracking problems and we believe that detecting those scenarios ahead of time is a possible use of the closed-loop enclosure. This is discussed in depth in the Discussion Chapter 5.2. The sequence of boxes $([y_n])_{n\in 0..N}$, and Picard boxes $([\tilde{y}_n])_{n\in 0..N-1}$ it outputs is guaranteed to enclose closed-loop trajectories because Algorithm 2 encloses optimal co-states at each time τ_n, and validated simulation encloses closed-loop Dynamics (2.10) which work like open-loop dynamics (2.9) (see Chapter 2 for details). The co-state output $([p_n])_{n\in 0..N-1}$ monitors the optimal co-state that the system finds when it recomputes its control.

Algorithm 4: Paving-based algorithm for the closed-loop enclosure.

Data: $[y_i]$, times of interest $t_i < \tau_1 < ... < \tau_{N-1} < t_f$
Result: $([y_n])_{n\in 0..N}$, $([p_n])_{n\in 0..N-1}$ enclosures at times τ_n, $([\tilde{y}_n])_{n\in 0..N-1}$, $([\tilde{p}_n])_{n\in 0..N-1}$ enclosures over $[\tau_n, \tau_{n+1}]$.

$[y_0] \leftarrow [y_i]$.
$n \leftarrow 0$.
while $n < N$ **do**
  $[y_{n+1}], [\hat{p}_n], [\tilde{y}_n], [\tilde{p}_n] \leftarrow \emptyset$.
  Split $[y_n]$ in a paving $([y_n]_k)_{k\in 0..K-1}$.
  $k \leftarrow 0$.
  while $k < K$ **do**
    $[\hat{p}] \leftarrow$ Algorithm 2 with $[y_n]_k$.
    $[\hat{p}_n] \leftarrow [\hat{p}_n] \cup [\hat{p}]$.
    Simulate System (2.10) from τ_n to τ_{n+1} with initial state $([y_n]_k, [p])$.
    $([y], [\hat{y}], [\hat{p}]) \leftarrow$ end state of simulation.
    $[y_{n+1}] \leftarrow [y_{n+1}] \cup [y]$.
    $[\tilde{y}_n], \leftarrow [\tilde{y}_n] \cup$ Picard boxes of simulation.
    $k \leftarrow k + 1$.
  end
  $n \leftarrow n + 1$.
end

The critical difference between closed-loop and anticipative enclosure is that at each recomputation time, we enclose the part of the current co-state set that satisfies the optimality condition, rather than compute a new one.

Indeed, as stated in Chapter 2, the trajectories of interest are optimal between the initial time and current time, meaning the true co-state lies in the purple set in Figure 3.7, which is the current co-state, obtained by simulating from the previous recomputation time. And they are also optimal between this time and the final time, meaning they also satisfy the optimality condition at the current time, hence lie in the blue set in Figure 3.7. Hence, Algorithm 6, which computes the anticipative enclosure, use the full Inflate&Contract Algorithm 2 only at the start to find the initial co-state. Afterward, it uses Contract Algorithm 5, which only contracts the box to find all solutions within the inputted initial search area, to refine the box of co-state obtained by integrating Dynamics 1.11 from preceding iterations. A paving is used to improve accuracy, as presented in Figure 3.7.

Algorithm 5: Computes a thin enclosure $[p_\tau]$ of the optimal co-states in the initial search area.

Data: $[y_\tau]$ and an initial searching area $[p_\tau]$.
Compute $\mathcal{C}_K([p_\tau])$ using Algorithm 1.
while $\mathcal{C}_K([p_\tau]) \neq \emptyset$ *and* $D_{Hausdorff}([p_\tau], \mathcal{C}_K([p_\tau])) > precision$ **do**
    $[p_\tau] \leftarrow \mathcal{C}_K([p_\tau])$.
    Compute $\mathcal{C}_K([p_\tau])$ using Algorithm 1.
if $\mathcal{C}_K([p_\tau]) = \emptyset$ **then**
    $[p_\tau] \leftarrow \emptyset$.

Because Algorithm 5 only does fixed point iteration it always converges. Hence Algorithm 6 terminates as long as Algorithm 2 does. The sequence of boxes $([y_n])_{n\in 0..N}$, $([p_n])_{n\in 0..N}$, and Picard boxes $([\tilde{y}_n])_{n\in 0..N-1}$, $([\tilde{p}_n])_{n\in 0..N-1}$ it outputs is guaranteed to enclose optimal trajectories because Algorithm 2 encloses initial co-states, validated simulation enclose all trajectories springing from them, hence all optimal trajectories as well as open-loop trajectories, and the later are culled by Algorithm 5 (see Chapter 2 for details).

Due to the paving complexity, these algorithms cannot be applied to high dimensional problems. Nonetheless, their result can be highlighted on low dimensional problems.

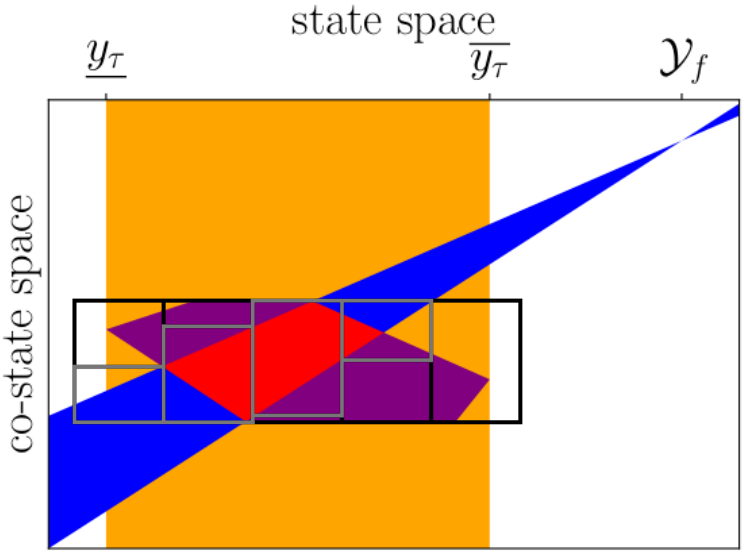


Figure 3.7: To enclose the red set, the box obtained by solving the OCP and simulating (see Figures 3.3 and 3.6) is paved along state space, yielding the black boxes. Algorithm 5 is applied to each box. It encloses the intersection between the box and the blue solution set and yields the gray boxes. Note that some boxes like the rightmost box might contain no solution, in which case Algorithm 5 outputs $\emptyset$. Precision could be enhanced by paving along the co-state space as well, thus better enclosing the purple set and then the red set. This would however double the dimension of the paving, hence square its complexity.

Algorithm 6: Paving-based algorithm for the anticipative enclosure.

Data: $[y_i]$, times of interest $t_i < \tau_1 < ... < \tau_{N-1} < t_f$.
Result: $([y_n])_{n\in 0..N}$, $([p_n])_{n\in 0..N}$ enclosures at times τ_n, $([\tilde{y}_n])_{n\in 0..N-1}$, $([\tilde{p}_n])_{n\in 0..N-1}$ enclosures over $[\tau_n, \tau_{n+1}]$.

$([y_n])_{n\in 0..N}, ([p_n])_{n\in 0..N}, ([\tilde{y}_n])_{n\in 0..N-1}, ([\tilde{p}_n])_{n\in 0..N-1} \leftarrow \emptyset$.
$[y_0] \leftarrow [y_i]$.
Split $[y_i]$ in a paving $([y_i]_k)_{k\in 0..K-1}$.
$k \leftarrow 0$.
while $k < K$ **do**
  $[p] \leftarrow$ Algorithm 2 with $[y_i]_k$.
  $[p_0] \leftarrow [p_0] \cup [p]$.
  Simulate System (1.11) from t_i to τ_1 with initial state $([y_i]_k, [p])$.
  $([y_1, p_1]) \leftarrow ([y_1, p_1]) \cup$ end state of simulation.
  $([\tilde{y}_0], [\tilde{p}_0]) \leftarrow ([\tilde{y}_0], [\tilde{p}_0]) \cup$ Picard boxes of simulation.
  $k \leftarrow k + 1$.
end
$n \leftarrow 1$.
while $n < N$ **do**
  $[y_{n+1}], [p_{n+1}], [\tilde{y}_n], [\tilde{p}_n] \leftarrow \emptyset$.
  Split $[y_n]$ in a paving $([y_n]_k)_{k\in 0..K-1}$.
  $k \leftarrow 0$.
  while $k < K$ **do**
    $[p] \leftarrow$ Algorithm 5 with $[y_n]_k$.
    $[p_n] \leftarrow [p_n] \cup [p]$.
    Simulate System (1.11) from τ_n to τ_{n+1} with initial state $([y_n]_k, [p])$.
    $([y_{n+1}, p_{n+1}]) \leftarrow ([y_{n+1}, p_{n+1}]) \cup$ end state of simulation.
    $([\tilde{y}_n], [\tilde{p}_n]) \leftarrow ([\tilde{y}_n], [\tilde{p}_n]) \cup$ Picard boxes of simulation.
    $k \leftarrow k + 1$.
  end
  $n \leftarrow n + 1$.
end

3.3 Application to a problem in low dimension

In this section, we apply algorithms of Section 3.2.2 to a double integrator with quadratic cost which is the main system of our first article [13] and a stepping stone toward aerospace problems. Then the limits of the algorithms are brought to light and the specifications of later methods are stated.

Consider a double integrator in a uniform gravity field and with a reactor thrust. It is subject to a quadratic continuous cost and a quadratic penalization on the final state. We chose this problem because it is used to initialize a continuation method to solve Goddard's problem [19, 20].

$$\min \int_0^{t_f} \frac{||u||^2}{2} dt + K_v \frac{||v(t_f)||^2}{2}, \\ + K_r \frac{||r(t_f) - r_f||^2}{2}, \quad \text{such that} \quad \begin{cases} \dot{r}(t) = v, \\ \dot{v}(t) = -\frac{C}{m} u - G e_2, \\ r(t_i) = r_i, \quad v(t_i) = v_i, \end{cases} \tag{3.6}$$

where G is the normalized gravity field, e_2 is the unit vector of the vertical axis, C is the maximum thrust and m is the mass of the system, which is assumed to be constant.

By using the method in Chapter 2, this OCP turns into the following two point boundary value problem:

$$\begin{cases} \dot{r}(t) = v, \\ \dot{v}(t) = -p_v \left(\frac{C}{m}\right)^2 - G e_2, \\ \dot{p_r}(t) = 0, \\ \dot{p_v}(t) = -p_r, \end{cases} \\ r(t_i) = r_i, \qquad v(t_i) = v_i, \\ p_r(t_f) - K_r r(t_f) = 0, \quad p_v(t_f) - K_v v(t_f) = 0. \tag{3.7}$$

The values used are inspired by the take-off problem in [19]. The parameter nominal values are the same as in [19]: $C = 3.5$, $G = 1$ and $b = 1$. The initial position of the system corresponds to the final position of the take-off mission. The initial velocity has been chosen to be coherent with a re-entry mission.

The solution of the nominal case is presented on Figure 3.8.

The initial position has an uncertainty of around 5 km, which is about 2% of the span of the trajectory. The initial speed has a relative uncertainty of 1.7% and the parameter ratio C/m has a relative uncertainty of 0.2%. However, the mission duration is greater than the optimal duration so as to emphasize the differences between the enclosures. The longer the mission, the more the system has to fight gravity, the more uncertainties are accumulated along the vertical axis.

Figures 3.9, 3.10 and 3.11 depict enclosures of the double integrator computed using Algorithms 3, 4 and 6.

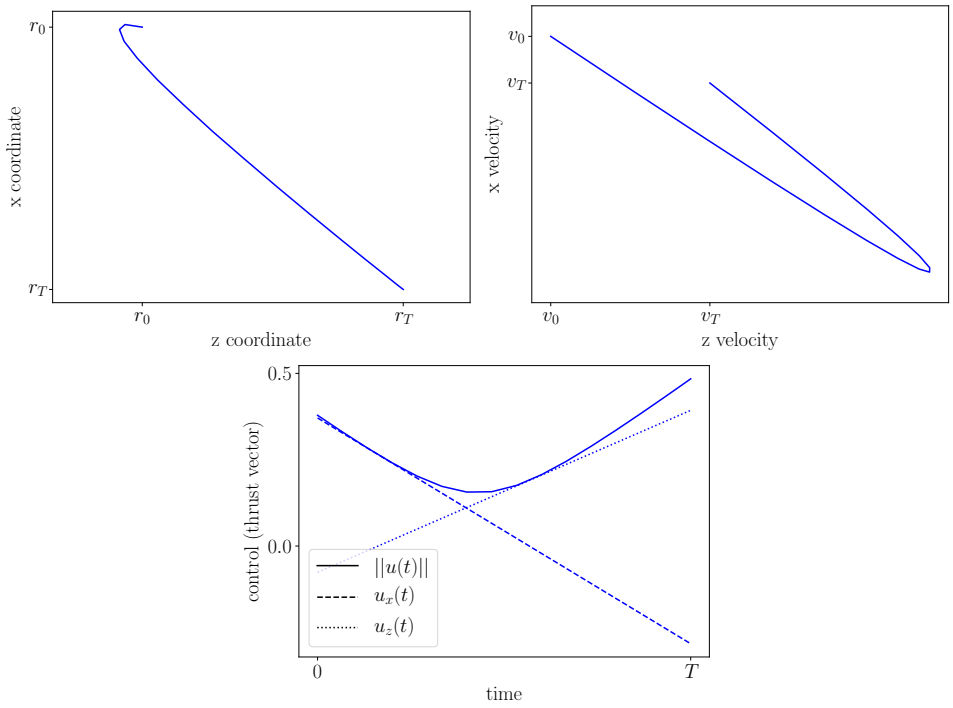


Figure 3.8: Optimal trajectory, velocity space trajectory and control for the a double integrator with quadratic cost.

Due to the system being in dimension 4, the paving complexity is high. To keep computation time low, we did not compute the resolvent with System (3.4). Instead, because System (3.7) is linear time invariant with a strictly triangular matrix, there are analytic formulae for the flow and the resolvents and we used them in Algorithm 1. Still, we observed that validated simulation yields the same resolvent as the analytical formula and are confident one would obtain the same result given enough computation power.

The anticipative enclosure on Figure 3.9 starts wide because of initial uncertainty but gets thinner as all the system converge to the target. As a result, the final box is very thin.

Contrarily, the open-loop enclosure on Figure 3.10 becomes wider over time and the final box is very big. Systems over-shoot or under-shoot the target because of their lack of correction.

Lastly, the closed-loop enclosure on Figure 3.11 is somewhere in between the two other enclosures. These systems deviate from their optimal trajectory but correct it, leading to a bulge in the middle and end of the trajectory that gets abruptly smaller at the end. To sum up, those three enclosures corroborate our observations from the analytical enclosures of the integrator in Section 2.5.

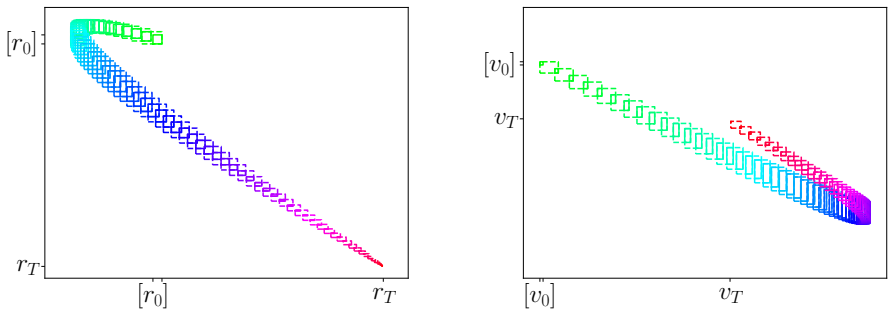


Figure 3.9: Anticipative enclosure of position and velocity.

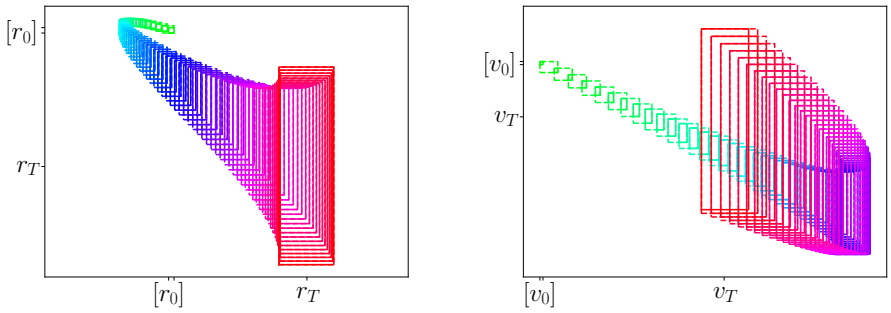


Figure 3.10: Open-loop enclosure of position and velocity.

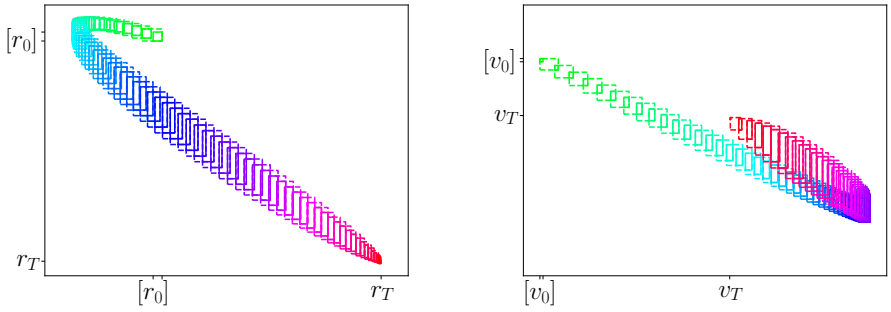


Figure 3.11: Closed-loop enclosure of position and velocity.

While being able to enclose a double integrator is encouraging, it is not quite the same as enclosing an aerospace problem, and the interval-based algorithms of this chapter are not up to the task for several reasons.

3.3.1 Limits of this approach

While these algorithms are a step toward the goal of computing the enclosures introduced in Chapter 2, we highlighted critical issues.

- The exponential complexity of paving forbids the resolution of high dimensional systems. This also excludes hybrid systems as each discrete event adds variables that need to be paved (see for instance [56]).
- The loss of correlation between state and co-state that happens in between OCP resolution and simulation causes wrapping effect and forces us to intersect our co-state with the solution set several times in Anticipative Algorithm 6 when ideally we would solve the OCP only once.
- Computing a resolvent considerably increases the size of the dynamical system, which considerably increases computation time and can cause validated simulation to fail return the entire vector space because it failed to enclose the system.

As a consequence, Chapter 4 focuses on avoiding paving, especially for hybrid discrete events, and we question whether computing a resolvent is necessary. To that end, Interval are replaced by another set representation.

Chapter 4

Replacing intervals by constrained symbolic zonotope

In this chapter, we develop methods for symbolic constrained zonotopes, to overhaul the algorithms of Section 3.2.2. Indeed, symbolic zonotopes keep linear correlations between all variables in the program, which addresses the issues highlighted in Section 3.3.1. This allows us to address the entire problem as highlighted in Chapter 2 which we recall here:

$$\begin{aligned} &\min_{u(\cdot)} \int_{t_i}^{t_f} \ell(y(t), u(t), \xi(t))dt + \Psi(y(t_f)), \\ &\text{s.t.} \begin{cases} \dot{y}(t) = f(y(t), u(t), \xi(t)), \\ y(t_i) \in \mathcal{Y}_i, \quad y(t_f) \in \mathcal{Y}_f, \quad t_f \text{ is free}, \\ c(y(t)) \leq 0. \end{cases} \end{aligned} \tag{4.1}$$

We first list our contributions to constrained symbolic zonotopes, we notably propose constrained spatio-temporal zonotopes to check for collisions with unsafe of guard set and enclose hybrid systems during events. We use those spatio-temporal zonotopes to build a Boundary Value Problem (BVP) solver, which we presented at CDC21 [14], then we propose a simpler and faster BVP solver that relies on the fundamental invariant of affine arithmetic [29]. The latter is used to compute the three enclosures of Chapter 2, and compared to the analytical OCP of Chapters 2 and to a simple nonlinear aerospace problem which opens the way to our most advanced results in Chapter 5.

4.1 Enhancing the BVP solver with constrained zonotopes

In this section, we use symbolic zonotopes [29], constrained zonotopes [63] and zonotope validated simulation library DynIbex [4] to build a BVP solver.

To that end, we propose constrained spatio-temporal zonotopes which are a convenient way to simulate hybrid systems. Then we use them to apply the propagate boundary conditions of our BVP, Lastly, we adapt the inflate & contract algorithm from Chapter 3.

While working with constrained zonotopes, we had several ideas to improve this formalism. We note constrained zonotopes $\mathbb{Z}^{\mathbb{A}} = \{\mathbf{Z}_0 + \mathbf{Z} \cdot \epsilon | \epsilon \in [-1,1]^{d_\epsilon}, \mathbf{A}_0 + \mathbf{A} \cdot \epsilon = 0\}$ to emphasize that the constraint part $\mathbb{A}$ is a zonotope on its own rather than a subpart like in our main inspiration [63]. Indeed, because we use symbolic zonotopes as defined in [29], noise symbols are shared by all zonotopes in the entire program, hence $\mathbb{Z}$ and $\mathbb{A}$ can be handled separately.

The main benefit is that a single zonotope $\mathbb{A}$ can be used as the constraint part of several zonotopes, even zonotopes corresponding to quantities that were computed at another point in the program. This property is derived from the fundamental invariant of affine arithmetic [29] and formalized as follow:

Proposition 1. *Consider a set $\mathcal{X}$, enclosed in a zonotope $\mathcal{X}$: $\mathcal{X} \subset \mathbb{X} \subset \mathbb{R}^d$; let a constraint function $c : \mathbb{R}^d \to \mathbb{R}^{d_c}$. The implicit set $\mathcal{X}^c$ defined as $\mathcal{X}^c = \{x \in \mathcal{X} : c(x) = 0\}$ is a subset of $\mathbb{X}^{\mathbb{A}}$, with $\mathbb{A} = [c](\mathbb{X})$.*

The zonotope $\mathbb{A} = [c](\mathbb{X})$ is the evaluation of a zonotope valued counterpart of function c on the zonotope $\mathbb{X}$, which was computed with affine arithmetic.

Proof. The fundamental invariant of affine arithmetic states that *at any stable instant in an affine arithmetic computation, there is a single assignment of values from $[-1,1]$ to each of the noise variables in use at the time that makes the value of every affine form equal to the value of the corresponding quantity in the ideal computation* [29]. Ideal computations refer to the mathematically perfect value one would obtain by doing the computation with infinite precision. Hence, if we take any $x \in \mathbb{X}$, there is an assignment of the noise values such that $\mathbb{X}$ and $\mathbb{A}$ are equal to their corresponding quantity, that are x and $c(x)$. This can be formulated as the follows:

$$\forall x \in \mathcal{X}, \quad \exists \epsilon \in [-1,1]^{d_\epsilon} \quad \text{s.t.} \quad \begin{cases} x & = \mathbf{X}_0 + \mathbf{X} \cdot \epsilon, \\ c(x) & = \mathbf{A}_0 + \mathbf{A} \cdot \epsilon. \end{cases}$$

From there, we conclude:

$$\forall x \in \mathcal{X}^c, \quad \exists \epsilon \in [-1,1]^{d_\epsilon} \quad \text{s.t.} \quad \begin{cases} \mathbf{X}_0 + \mathbf{X} \cdot \epsilon = & x, \\ \mathbf{A}_0 + \mathbf{A} \cdot \epsilon = & 0. \end{cases}$$

By definition, $\forall x \in \mathcal{X}^c, x \in \mathbb{X}^{\mathbb{A}}$. □

This property makes the combination of symbolic zonotopes and constrained zonotope very convenient. By contrast, constrained zonotopes in [63] are not symbolic and that arises the need to put coordinates, constraints and any other affine form of interest in one big zonotope and drag all these affine forms along in every operation. The ability to apply constraints at one point of the program to a set at another point is particularly useful for solving BVPs, as boundary constraint on the final zonotope can be added on the initial zonotope through the noise symbols.

However, to retain correlations between zonotopes, operations need to be done with Affine Arithmetic [29], which means using zonotopes exclusively. In particular, this means we cannot simulate the discrete events of hybrid systems by switching to a more adapted state representation as in [7, 8, 24]. As a consequence we propose a method to simulate hybrid systems using constrained zonotopes.

4.1.1 Developing spatio-temporal zonotopes for simulation of hybrid systems

The validated simulation used by DynIbex [4] encloses possible trajectories in sequences of zonotopes and Picard boxes, with zonotopes enclosing the system at instants and Picard boxes enclosing the system over a time range. This causes two difficulties, which are illustrated in Figure 4.1. Firstly, it is hard to assert the safety with respect to state constraints because those constraints must be satisfied not at given instants but at all times. Since zonotopes only enclose the system at instant, the only way to ensure the satisfaction of state constraints is proving that all Picard boxes stay out of the unsafe set, and that can cause a false positive due to wrapping effect. Secondly, enclosing a hybrid system through a discrete event accurately is challenging. Indeed, if the event time is variable, then no single zonotope will enclose the system at all possible event time, so one must apply the jump to a Picard box instead, which causes significant over approximation. There are techniques to alleviate this, such as subdividing the event time range and simulating systems that have an event over each sub time interval separately (see [56, 24]) or switching to other set representations to simulate the event with more accuracy (see [24, 7, 8]). However, those incur significant computational costs, and crucially they invalidate the symbolic zonotope approach because they require switching back and forth between zonotopes and other set representations, and those switch create new zonotopes with new noise symbols that have no correlation with past zonotopes in the program.

To alleviate the shortcomings of the usual validated simulation framework, we introduce spatio-temporal zonotopes.

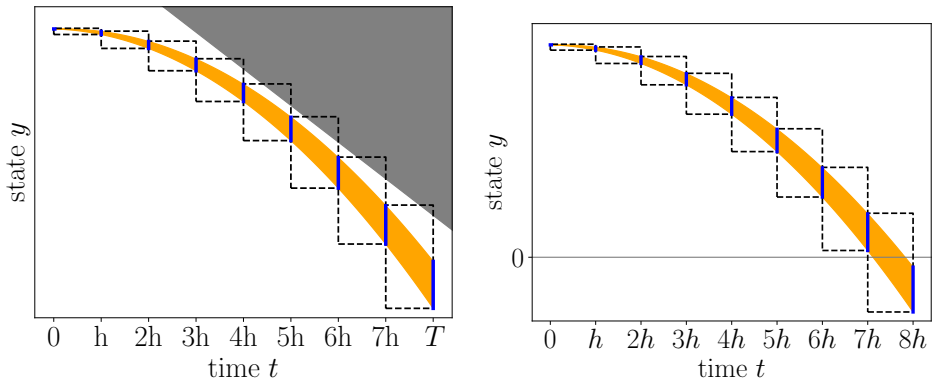


Figure 4.1: Difficulties of Validated simulation. The set of possible trajectories of a falling ball with unknown initial velocity are computed analytically (orange) and enclosed with DynIbex [4]. Left: the safety of the system is not guaranteed because boxes intersects with the gray unsafe set due to wrapping effect. Right: the intersection with the black guard set is loosely enclosed in a Picard box because the blue zonotopes all line up with the initial time rather than event time.

A spatio-temporal zonotope is of the form:

$$\mathbb{Z} = \begin{pmatrix} \mathbb{T} \\ \mathbb{Y} \end{pmatrix},$$

where $\mathbb{T}$ is a one dimensional affine form that encloses a time range $[t, t+h_Z]$, with h_Z is a duration chosen by the user, and $\mathbb{Y}$ is a zonotope enclosing the state on time range $[t, t + h_Z]$. Such a zonotope can be built using the following procedure which is illustrated in Figure 4.2. We start with a given zonotope $\mathbb{Y}_0$ enclosing the state at instant $t = 0$, either because it is an input of the problem or because it is the output of a preceding validated algorithm. We compute a Picard box $[\tilde{y}]$ such that $\forall s \in [0, h_Z], y(s) \in [\tilde{y}]$ using validated simulation. Then a validated Taylor interpolation [47] is used. We use the following order zero interpolation (4.2).

$$\forall t \in [0, h_Z], \quad y(t) \quad \in \mathbb{Y}_0 + t \times [f]([\tilde{y}]). \tag{4.2}$$

This inclusion can be derived similarly to Picard's operator:

$$\begin{array}{lll} \forall t \in [0, h_Z], & y(t) & = y(0) + \int_0^t f(y(s))ds \\ & & \in \mathbb{Y}_0 + \int_0^t [f]([\tilde{y}])ds \\ & & \in \mathbb{Y}_0 + t \times [f]([\tilde{y}]). \end{array}$$

It follows from Equation (4.2) that the spatio-temporal zonotope Z can be initialized with Formula (4.3).

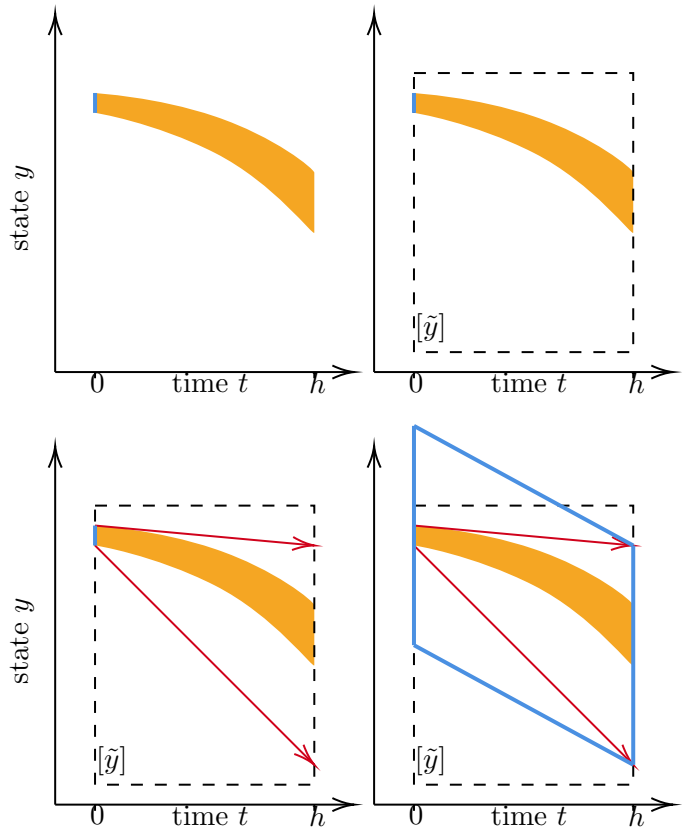


Figure 4.2: Building a spatio-temporal zonotope. i) Start with an enclosure $\mathbb{Y}_0$ at time 0. ii) Compute Picard box $[\tilde{y}]$ that encloses all trajectories spanning from $\mathbb{Y}_0$ for $t \in [0, h]$ with Picard's operator. iii) Enclose dynamics on $[\tilde{y}]$. iv) Deduce a zonotope that encloses trajectories over the range $[0, h]$. This differs from regular validated simulation in that the last step encloses the system on the entire range rather than h only, allowing to have linear correlations between space and time.

$$\mathbb{Z}_0 = \begin{pmatrix} \mathbb{T} \\ \mathbb{Y}_0 + \mathbb{T} \times [f]([\tilde{y}]) \end{pmatrix}, \tag{4.3}$$

where $\mathbb{T} = \frac{h_Z}{2} + \frac{h_Z}{2}\epsilon_0$ with $\epsilon_0 \in [-1, 1]$ is a one dimensional zonotope enclosing time range $[0, h_Z]$. We believe that there is considerable room for improvement with this formula. Indeed, the gap between the tube of trajectories and the zonotope is significant, as can be seen in Figure 4.2. We switch to a centered formula in later sections as it provides a tighter enclosure (see Figure 4.3) but it could be improved further with higher order methods.

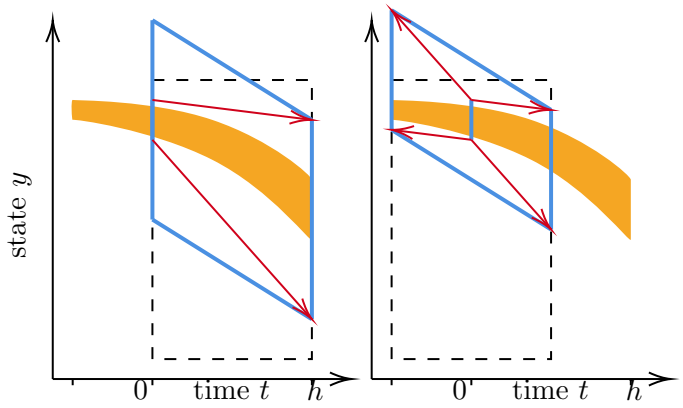


Figure 4.3: Forward (left) and centered (right) method to building a spatio-temporal zonotope. Forward goes forward in time from its initial enclosure while centered is centered on time 0. For a same Picard box, centered method gives a tighter zonotope.

Once initialized, spatio-temporal zonotope may be simulated with dynamics:

$$\begin{pmatrix} \dot{t} \\ \dot{y} \end{pmatrix} \in \begin{pmatrix} 1 \\ [f]\,(y, [\xi]) \end{pmatrix}, \tag{4.4}$$

which yields an enclosure of all possible trajectories. The entire procedure to enclose trajectories at all time is summed up in Algorithm 7 and its result is represented in Figure 4.4.

This algorithm initializes spatio-temporal zonotopes by computing a Picard box with validated simulation, then applying Formula (4.3). Then it simulates forward.

As illustrated in Figure 4.4, if the integration step h is equal (or inferior) to the duration h_Z covered by each zonotope, then the possible trajectories are contained in the sequence of spatio-temporal zonotopes. These zonotopes can be used to validate a state constraint, for instance staying out of an unsafe set as illustrated in Figure 4.4. This allows to enclose the state during the discrete, then refine it with a constraint and use this constrained zonotope to initialize a new simulation after the discrete event, thus simulating a hybrid system.

This is illustrated on a bouncing ball formalized as the automaton in Figure 1.1 in Section 1.1.2. It is subject to uncertainties on the bounce parameter: $\lambda \in [\lambda]$, and uncertainties on the initial velocity: $v_i \in [v_i]$.

Algorithm 7: Enclose all possible trajectories in spatio-temporal zonotopes

Result: $(Z_n)_{n\in 0..N}$ a sequence of zonotopes that encloses the system's state between $t = t_i$ and $t = t_f$

Data: The final time t_f, a duration h_Z, an enclosure of the initial state $\mathbb{Y}_i$, dynamics f

$n \leftarrow 0$.

$\mathbb{T} \leftarrow \frac{h_Z}{2} + \frac{h_Z}{2}\epsilon_0$.

$[\tilde{y}] \leftarrow$ Picard box (or union of Picard boxes) of the validated simulation of dynamics f with initial state Y_0 and horizon h_Z.

$Z_0 \leftarrow (\mathbb{T}; \mathbb{Y}_0 + \mathbb{T} \times [f]([\tilde{y}]))$.

while $\max Z_{n,0} < t_f$ **do**

  $n \leftarrow n + 1$.

  $Z_n \leftarrow$ the validated simulation of extended dynamics (4.4) with initial state Z_{n-1} and horizon $h = h_Z$.

end

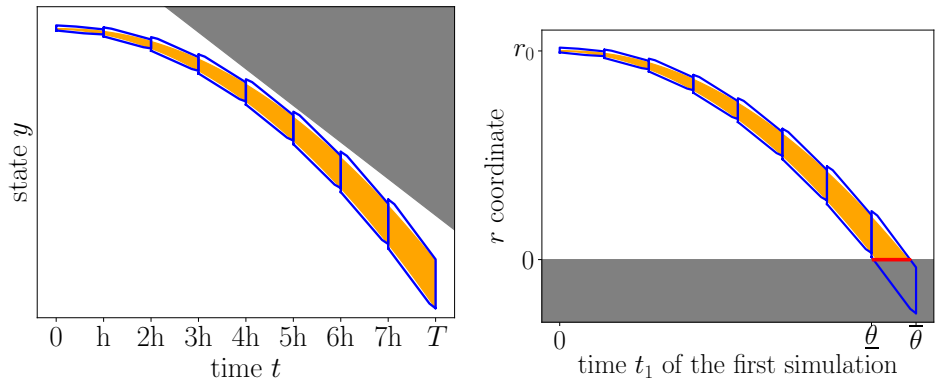


Figure 4.4: The challenges of Figure 4.1 are addressed with spatio-temporal zonotopes. Left: zonotopes enclose the tube more tightly than boxes and guarantee that the system does not enter the unsafe set. Right: the intersection with the guard set is enclosed in a single zonotope, and refined with a constraint, yielding the red constrained zonotope.

We recall the bouncing ball dynamics:

$$\begin{cases} \dot{r}(t) = v(t) \\ \dot{v}(t) = -9.81 \\ r(0) = 1 \\ v(0) \in [v_0], \end{cases}$$

and its state dependent event:

$$\text{if } \psi(r(t_1^-)) = r(t_1^-) = 0 \text{ then } v(t_1+) \in -[\lambda]v(t_1-).$$

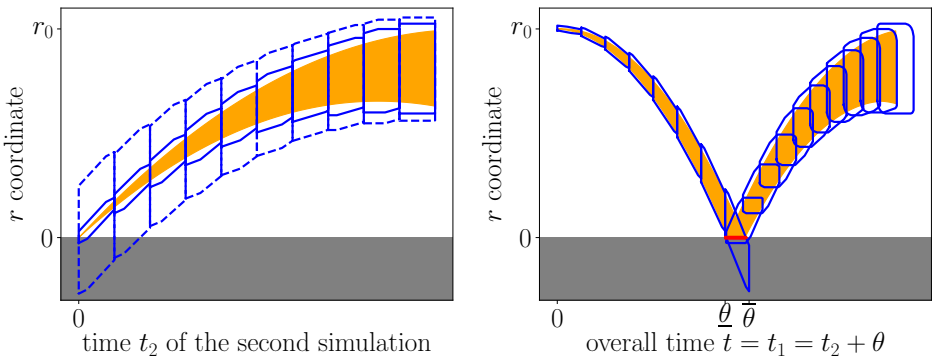


Figure 4.5: In Figure 4.4 a bouncing ball is simulated until it bounces on the ground. Its last zonotope is used to initialize a simulation after bouncing, yielding the enclosure on the left. The entire trajectory is displayed on the right. The change of shape is due to the fact that the second simulation starts at bounce time rather than initial time, and the duration between the two is variable. The dashed hull is the enclosure without taking into account that the ball is on the ground at bounce time, while plain has this information in the form of a zonotope constraint.

A preliminary validated simulation is done to find bounds on event time. For instance, on the right of Figure 4.1, it can be seen that at time $7h$ the ball enclosure is above the ground and at time $8h$ it is below, hence the event is guaranteed to happen in $t \in [7h, 8h]$. Then Algorithm 7 is applied. This yields the enclosure displayed on the right of Figure 4.4. Then the last zonotope of that simulation is used to initialize another simulation with Algorithm 7 to compute the post bounce trajectory, which is displayed on the left of Figure 4.5. The entire trajectory is showcased on the right of Figure 4.5. It encloses the tube of possible trajectories in a single forward swipe.

These methods can enclose trajectories of a hybrid system in a sequence of constrained zonotopes without ever switching to another set representation and lose correlation. Next section will use that to enclose solutions of BVPs.

4.1.2 Propagating boundary constraints backward

In this section, we consider our full hybrid BVP. We have hybrid system:

$$\begin{cases} \dot{x}(t) = g_n(x(t), \xi(t)), & \forall t \in [t_{n-1}^+, t_n^-] \\ x(t_n^+) = j_{n+1}(x(t_n^-)), & \forall n \in 0..N-1 \\ x(0) = x_0 \end{cases} \tag{4.5}$$

where $0 = t_0 < t_1 < ... < t_N = t_f$ are transition times and $j_n : \mathbb{R}^{1+2d} \mapsto \mathbb{R}^{1+2d}$ are jump functions. It is subject to switching and optimality conditions:

$$C_n(x(t_n^-)) = 0, \forall n \in 1..N. \tag{4.6}$$

Our first attempt at enclosing trajectories in zonotopes followed naturally the same reasoning as with intervals. We linearize the system at several times by computing a resolvent of the system and using a formula similar to that of our Krawczyk contractor in Chapter 2. Our second attempt uses properties of affine arithmetic to compute a tube in a single forward swipe.

First approach: linearization of the flow with a resolvent

Our first method to propagate boundary constraints computes the first derivative of the BVP with affine arithmetic, then adds it as constraints using constrained zonotope formalism. This takes the form of the following equations, which are an affine arithmetic counterpart of the first derivative equation proposed in Section 2.3.

$$\mathbb{A} = \begin{pmatrix} [\nabla C_1]\,(x_1) \cdot \mathbb{W}_1 \\ [\nabla C_2]\,(x_2) \cdot \mathbb{W}_2 \\ \vdots \\ [\nabla C_N]\,(x_N) \cdot \mathbb{W}_N \end{pmatrix},$$

with for every n, $\mathbb{W}_n = [\mathbf{R}_{t_{n-1},t_n}](x_{n-1}) \cdot [\nabla j_{n-1}](x_{n-1}) \cdot \mathbb{W}_{n-1}$. x_{n-1}^- and $[\mathbf{R}_{t_{n-1},t_n}]$ are the solution at time t_{n-1} of System (4.7) and $x_n^+ = [j_n](x_n^-)$.

$$\begin{cases} \dot{x}(t) & \in [g_n]\,(x, [\xi]) \\ \dot{\mathbf{R}}_{t_{n-1},t}(x_i) & \in \left[\dfrac{\partial g_n}{\partial x}\right](x, [\xi])) \cdot \mathbf{R}_{t_{n-1},t}(x_i) \\ x(t_{n-1}) & = x_{n-1}^+ \\ \mathbf{R}_{t_{n-1},t_{n-1}}(x_n) & = I_d. \end{cases} \tag{4.7}$$

$\mathbb{W}_n$ are the derivatives with respect to the initial state and co-state, hence we also add the derivative with respect to switch time:

$$\forall i \in 1..N, \mathbb{A} \leftarrow \mathbb{A} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \nabla C_i(x_i) \cdot [g_n](x_i^-, [\xi]) \\ \nabla C_{i+1}(x_{i+1}) \cdot \mathbb{W}_{i+1}^{t_i} \\ \vdots \\ \nabla C_N(x_N) \cdot \mathbb{W}_N^{t_i} \end{pmatrix},$$

with $\mathbb{W}_i^{t_i} = [\nabla j_{i-1}](x_i^-) \cdot [g_i](x_i^-, [\xi]) - [g_{i+1}](x_i^+, [\xi])$ and $\mathbb{W}_n^{t_i} = [\mathbf{R}_{t_{n-1},t_n}](x_n) \cdot [\nabla j_{n-1}](x_n) \cdot \mathbb{W}_{n-1}^{t_i}, \forall n > i$.

We then build a method that computes these flows and resolvents with validated simulation as with Algorithm 1 and adds them as constraints, thus enclosing optimal co-states and switch times. With repeated computation of these constraints, optimal trajectories are enclosed.

This method however proved inefficient. As our grasp of constrained zonotopes and symbolic zonotope tightened, we realized that the process of validated simulation computes a linear correlations between initial and final state through the noise symbols, and that the resolvent is redundant. This lead us to a simpler and more precise method.

Second approach: propagation through noise symbols

This section applies the results of Section 4.1 to BVPs. The solution set of our BVP is defined implicitly as:

$$\mathcal{X}^C = \{(y_0, p_0, (t_n)) \in \mathcal{Y}_0 \times \mathbb{R}^d \times \mathbb{R}^N : C_n(\Phi_{0,t_n}(y_0, p_0)) = 0, \forall n\}.$$

Let a zonotope $\mathbb{P}_0$ of initial co-states and a zonotope $\mathbb{T}$ of switch times, and let $\mathbb{A} = [C_n](\mathbb{X}_n)$, where $\mathbb{X}_n = [\Phi_{0,\mathbb{T}_n}](\mathbb{Y}_0, \mathbb{P}_0)$ is an enclosure at time t_n computed with validated simulation and spatio-temporal zonotopes. By application of Proposition 1, all optimal co-states within $\mathbb{P}_0$ are enclosed in $\mathbb{P}_0^{\mathbb{A}}$ and optimal switch times within $\mathbb{T}$ are enclosed in $\mathbb{T}^{\mathbb{A}}$.

In less abstracts terms, Affine Arithmetic as presented in [29] is such that noise symbols keep correlations between all zonotopes in the program. In the case of validated simulation, this means that all state enclosures are correlated, in particular, enclosures at boundary times that are subject to optimality constraints are correlated to state enclosures at all other times. Hence, adding optimality condition as constraints on those boundary zonotopes also adds them on all other state enclosure through the noise symbols. This phenomenon is illustrated in Figure 4.6.

We propose a BVP Solver 8 that takes initial co-states and switch times in zonotope form and computes boundary constraints $\mathbb{A}$.

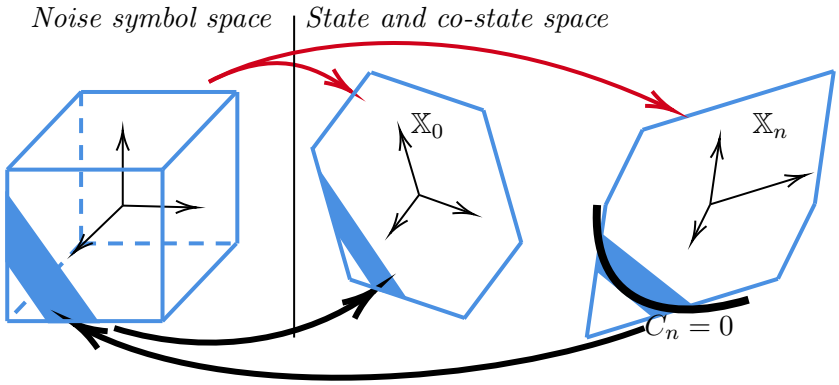


Figure 4.6: The boundary state $x(t_n)$ is enclosed in $\mathbb{X}_n$ and subject to $C_f(x(t_n)) = 0$. We compute $\mathbb{A} = [C_n](\mathbb{X}_n)$ and add it as linear constraints on the noise symbol hypercube. This yields constrained zonotope $\mathbb{X}_n^{\mathbb{A}}$ (plain blue) which conservatively enclose the part of the final zonotope that satisfies $C_f = 0$. Since all symbolic zonotopes are affine transformations of the same noise symbol hypercube (as shown by the red arrows), constraints on the hypercube can be applied to them as well. Hence, a constraint on $\mathbb{X}_n$ can be propagated to the initial zonotope $\mathbb{X}_0$ through the noise symbols as illustrated by the black arrows.

The result of this solver is illustrated on the integrator OCP with fixed final time in Figure 4.7. When boundary times are not fixed, we use spatio temporal zonotopes to enclose the system at boundary times (Line 7). BVP Solver 8 is different from Algorithm 7 in that it is not designed to enclose the system at all times. As a consequence, we simulate forward first and initialize our spatio-temporal zonotope afterward. This yields a tighter enclosure of the state at switch time. Indeed, a spatio-temporal zonotope that encloses the system over a duration is typically bigger than the initial zonotope it was built with. Since wrapping effect grows with the size of the zonotope, it is better to do the bulk of the simulation with that initial zonotope first and apply our formula afterward. The input $\mathbb{T}$ in Solver 8 is a temporal zonotope, with one coordinate for each switch time. Solver 8 returns $\mathbb{A}$ such that $\mathbb{P}_0^{\mathbb{A}}$ encloses all optimal co-states in $\mathbb{P}_0$ and $\mathbb{T}^{\mathbb{A}}$ encloses all switch times in $\mathbb{T}$, hence Section 4.1.3 embeds this solver in a method that provides $\mathbb{P}_0$ and $\mathbb{T}$ containing all solutions.

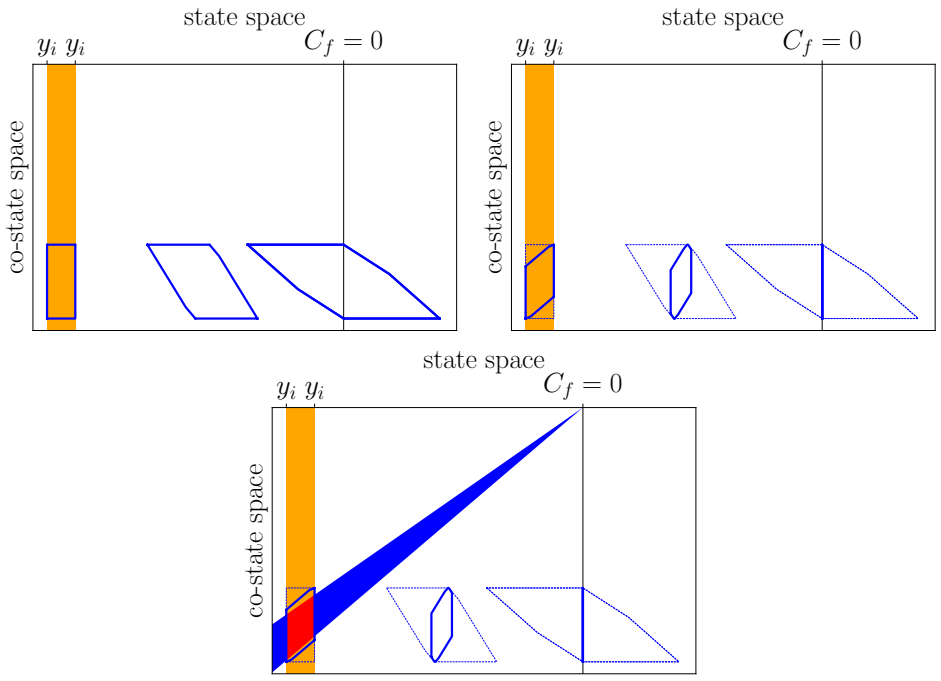


Figure 4.7: Result of constraint propagation on the integrator OCP. On the left, we illustrate Algorithm 8: we take an initial enclosure of state and co-state, we simulate up to the final time (from left to right) and compute $\mathbb{A} = [C_f](\mathbb{X}_f)$. On the right, we add constraint $\mathbb{A}$ as constraints on the noise symbols, which propagates to all zonotopes as illustrated on Figure 4.6. This yields an enclosure of final state and co-state couples that intersect with the solution set $C_f = 0$ (black line), and also an enclosure of initial state and co-state couples that are solution of the BVP. This initial constrained zonotope encloses the analytical solution set in red at the bottom. Note however that this initial constrained zonotope is a subset of the inputted initial enclosure, hence, it is necessary that the input of Solver 8 contains all solutions.

Algorithm 8: Computes boundary constraints as constrained zonotopes.

Data: $\mathbb{Y}_0$, $\mathbb{P}_0$, switch times $\mathbb{T}$.
Result: Boundary constraints $\mathbb{A}$.

$\mathbb{A} \leftarrow$ zonotope with 0 lines.
$\mathbb{X} \leftarrow (0, \mathbb{Y}_0, \mathbb{P}_0)$.
$n \leftarrow 0$.
while $n <$ *number of coordinate of* $\mathbb{T}$ **do**
  $\mathbb{X} \leftarrow$ validated simulation of dynamics g_n of System (2.4) over duration $\mathrm{mid}\,(\mathbb{T}_n)$.
  $[\tilde{x}] \leftarrow$ Picard box of system over time range $[\min(\mathbb{T}_n), \max(\mathbb{T}_n)]$.
  $\mathbb{X} \leftarrow \mathbb{X} + (\mathbb{T}_n - \mathrm{mid}\,(\mathbb{T}_n)) \times [g_n]([\tilde{x}])$.
  $\mathbb{A} \leftarrow (\mathbb{A}; C_n(\mathbb{X}))$.
  $n \leftarrow n + 1$.
end

Algorithm 8 is a forward simulation and some affine operations, hence it will terminate. The results of this method depends on the amount of linear correlation between zonotopes, which depend on how nonlinear the dynamic system is. It will typically struggle with highly nonlinear systems, like those encountered in Chapter 5. Nevertheless, we found in our experiments in Section 4.2 that it produces tighter results than the resolvent-based method of Section 4.1.2 while also being significantly faster.

This propagation of constraints allows the re-implementation of Inflate & Contract Algorithm 2.

4.1.3 Inflate & Contract Method with constrained zonotopes

There are no zonotope-based inflate & contract methods in the literature to the best of the author's knowledge, which is expected considering zonotope are not closed under intersection, rendering contraction difficult.

As a side note, we entertained the thought of engineering a constrained-zonotopes-based inflate & contract methods, as constrained zonotopes are closed under intersection. We came to the conclusion that it would fare worse than their interval counterparts for two reasons.

Firstly because it is hard to guarantee that a constrained zonotope is included in the interior of another constrained zonotope. Indeed, this requires checking that the constrained part does not touch its hull at any point. Computing vertices is very costly hence we tried proving strict inclusion on the noise box instead. Indeed, it can be deduced from the definition that if all noise symbols in $\mathbb{X}^{\mathbb{A}}$ take their value in $]-1, 1[$ then $\mathbb{X}^{\mathbb{A}}$ is included in $\mathbb{X}$. However, this approach is more pessimistic the more generators $\mathbb{X}$ has.

For instance if $\mathbb{X}$ has a single generator, and we have a constraint that translates as $3\epsilon_0 \in [-2, 2]$, then we have $\epsilon \in]-1, 1[$, but if we have three generators and constraint $\epsilon_0 + \epsilon_1 + \epsilon_2 \in [-2, 2]$ then no information can be deduced. To counteract this issue, one must take as many generators as there are variables, which is equivalent to taking a rotated box.

Secondly zonotope intersections in the form of constrained zonotopes have as much wrapping effect as their parent zonotopes, rendering contraction useless. Indeed, when representing the intersection of two zonotopes as a constrained zonotope $\mathbb{X}^{\mathbb{A}}$ using formulae in [63], the unconstrained hull $\mathbb{X}$ is one of the parent zonotope, and all affine arithmetic operations are done with this hull rather than the constrained part. Moreover, intersections add loads of constraints and noise symbols.

Instead, zonotopes will be used as a medium for a regular interval-method by defining a contractor with Algorithm 9.

Algorithm 9: Computes $\mathcal{C}([p_\tau])$ and $\mathcal{C}([t])$.

Data: $[y_\tau]$, $[p_\tau]$, switch times $[t]$.
Result: $\mathcal{C}([p_\tau])$, $\mathcal{C}([t])$.
1 Convert box $[p_\tau]$ to a zonotope $\mathbb{P}$.
2 Convert box $[t]$ to a zonotope $\mathbb{T}$.
3 Compute constraint part $\mathbb{A}$ with Algorithm 8.
4 $\mathcal{C}([p_\tau]) \leftarrow$ bounding box of $\mathbb{P}^{\mathbb{A}}$.
5 $\mathcal{C}([t]) \leftarrow$ bounding box of $\mathbb{T}^{\mathbb{A}}$.

As seen in Section 4.1.2, $\mathbb{P}^{\mathbb{A}}$ encloses all co-states in $\mathbb{P}$, hence in $[p_\tau]$. The bounding box in Line 4 is computed with a simplex solver and rounded outward with Neumaeir Scherbina post-processing [50], this ensures that all solution in $\mathbb{P}^{\mathbb{A}}$ are in $\mathcal{C}([p_\tau])$. The same holds for switch times. Hence, we have inclusion of our solution set. This procedure uses validated simulation and linear programming, hence it will terminate as long as validated simulation does.

The contractor outputted by Algorithm 9 is used in place of the Krawczyk contractor in Algorithm 2. This allows us to compute enclosures for aerospace problems.

4.2 Application

In this section we apply our new BVP solver to two systems. we first go back to the integrator with quadratic cost of Section 2.5, to compare our results with analytical solutions. Then we tackle a simple nonlinear hybrid aerospace problem.

4.2.1 Integrator with quadratic cost

For our first application, we recall the OCP of the integrator with quadratic cost of Section 2.5:

$$\min_{u(\cdot)\in\mathcal{U}} \int_{\tau}^{t_f} \frac{u^2}{2} dt \text{ s.t. } \begin{cases} \dot{y}(t) \in [\xi(t)]u, \\ y(t_i) \in [y_i], \\ y(t_f) = y_f, \\ t_f \text{ is fixed.} \end{cases} \tag{4.8}$$

As well as its BVP form:

$$\begin{cases} \dot{y}(t) \in [\xi(t)]^2 p, \\ \dot{p}(t) = 0, \\ \quad y(t_i) \in [y_i], \quad p(t_i) \text{ is free}, \\ \quad y(t_f) = y_f, \quad p(t_f) \text{ is free}. \end{cases}$$

We use the following uncertainties: $[y_i] = [-0.1, 0.1]$ and $[xi] \in [9.5, 10.5]$.

We first apply our inflate & contract algorithm with constrained zonotope contractor, its result is showcased on Figure 4.9.

We then compute the anticipative and closed-loop enclosures, which are displayed on Figure 4.8. We took unknown but constant parameters, which is why the anticipative enclosure is cone-shaped (see Section 2.5).

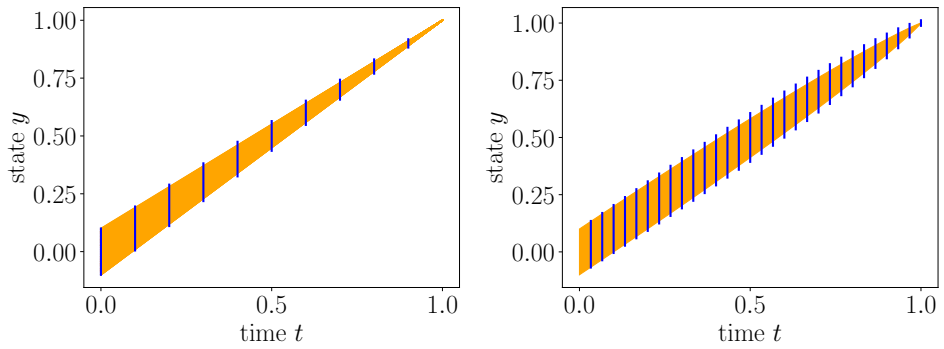


Figure 4.8: Analytical anticipative enclosure (orange, left) and closed-loop enclosure (orange, right) enclosed by constrained zonotopes. Constrained zonotopes successfully enclose the analytical set.

Now that we checked that our new constrained zonotope methods match with analytical solutions, we apply them to a nonlinear system with hybrid behaviors.

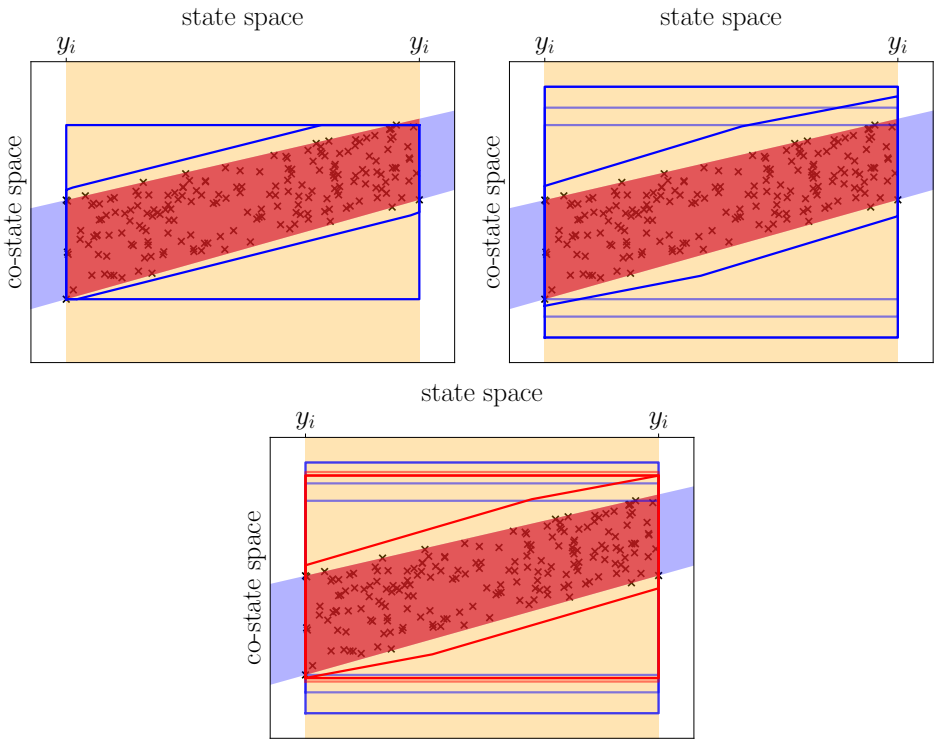


Figure 4.9: Inflate & contract algorithm with constrained zonotope contractor enclosing the integrator with quadratic cost, compared with the analytical solution set. The diamond sets are constrained zonotopes obtained by propagating boundary condition backward. Pale orange, pale blue and pale red are respectively the analytical set of possible (state, co-state) couples, the analytical set of optimal (state, co-state) and analytical solution set, as in Figure 2.1 and other figures throughout Chapters 2 and 3. i) The algorithm is initialized with the bounding box of Monte Carlo solutions (crosses), ii) inflated until the constrained zonotope fits vertically (which signals that the initial co-state is enclosed) iii) contracted. Constrained zonotopes always enclose the analytical solution set as expected.

4.2.2 Simple take-off problem

We illustrate our methods on the following vertical take-off problem:

$$\min_{u(\cdot)} \int_0^{t_f} |u| dt$$
$$\text{s.t.} \begin{cases} \dot{r}(t) = v(t), & r(0) = r_0, & r(t_f) = r_f, \\ \dot{v}(t) = -\frac{G}{r^2} + \frac{C}{m}u, & v(0) = v_0, & v(t_f) \text{ free}, \\ \dot{m}(t) = -b|u(t)|, & m(0) = m_0, & m(t_f) \text{ free}, \end{cases} \tag{4.9}$$

where r, v, m are position, velocity and mass of a launcher with a controlled thruster subject to gravity. It is the one-dimensional version of Goddard's Problem as formulated in [19], without the drag force. Application of the PMP yields:

$$\begin{array}{l} H(y, p, u, \xi) = p_r v + p_v(-\frac{G}{r^2} + \frac{C}{m}u) - p_m b|u|, \\ \dot{p_r}(t) = -2p_v(t)\frac{G}{r^3}, \\ \dot{p_v}(t) = -p_r(t), \\ \dot{p_m}(t) = p_v(t)\frac{C}{m^2}, \\ u = \begin{cases} \frac{p_v}{|p_v|} & \text{if } C|p_v| - (1 + bp_m)m > 0, \\ 0 & \text{if } C|p_v| - (1 + bp_m)m < 0. \end{cases} \end{array} \tag{4.10}$$

The expression of u in System (4.10) leads to a hybrid automaton with a free fall mode when $C|p_v| - (1 + bp_m)m > 0$, a full throttle mode when $C|p_v| - (1 + bp_m)m < 0$ and a singular mode if $C|p_v| - (1 + bp_m)m = 0$ over a period of time. It follows that $C|p_v| - (1 + bp_m) = 0$ at each mode transition. Further analysis of the problem shows that the optimal solution of the take-off problem with no drag force has $N = 2$ phases, starting by full throttle and ending in free fall. This coupled with transversality condition leads to constraints functions (4.11).

$$\begin{array}{rl} C_1(x) = & C|p_v| - (1 + bp_m)m, \\ C_2(x) = & \begin{pmatrix} r - r_f \\ p_v \\ p_m \\ v \end{pmatrix}. \end{array} \tag{4.11}$$

We took an exact initial state and uncertain parameters: $r_0 = 1$, $r_f = 1.01$, $v_0 = 0$, $m_0 = 1$ and $C \in [3.4, 3.6]$, $G \in [0.99, 1.01]$ and $b \in [6.8, 7.2]$. We computed 100 optimal trajectories with parameters taken at random in their respective intervals.

Figure 4.11 shows the workings of Inflate & Contract Algorithm with the Constrained Zonotope Contractor 9. We see it successfully encloses Monte Carlo results.

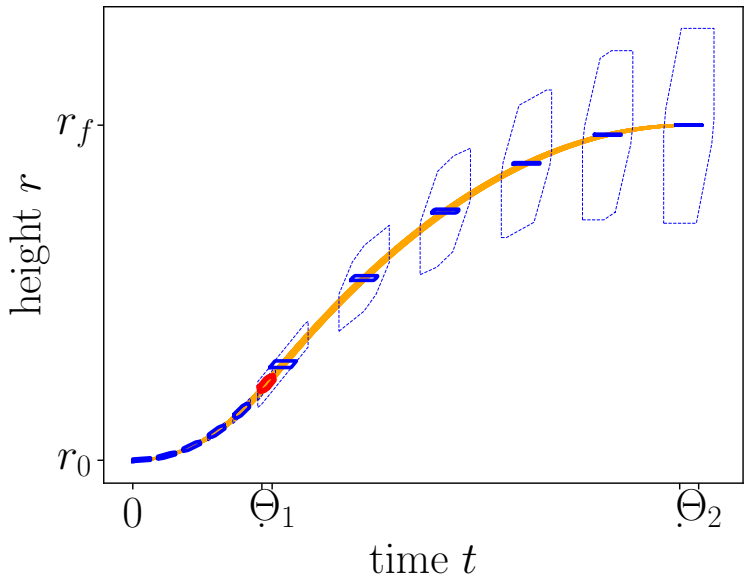


Figure 4.10: Results on Goddard's Problem (4.9). Orange: simulations with random parameters. t_1 has two graduations that indicate bounds $\underline{t}_1$ and $\overline{t}_1$, likewise for $t_f = t_2$. The generated constrained zonotopes (blue, plain) are a much more precise enclosure of optimal trajectories than the unconstrained hull (blue, dashed) which accumulates error. The red zonotope encloses the system during the transition from full throttle to free fall.

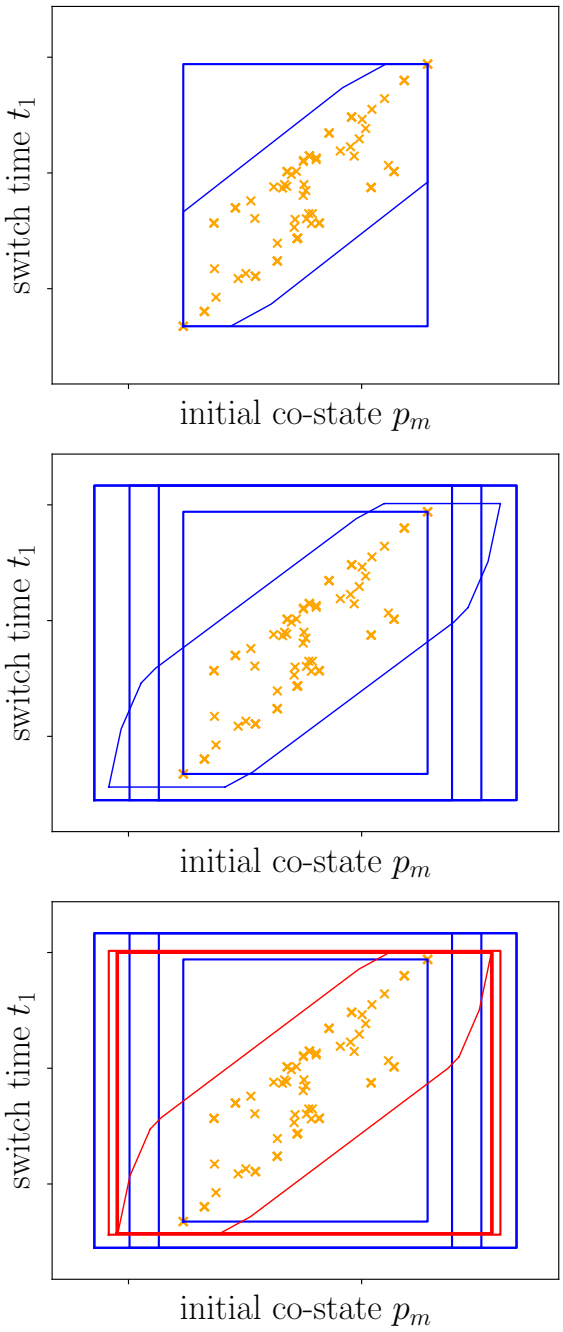


Figure 4.11: Inflate & Contract with constrained zonotopes on Goddard's problem. The search box is 5 dimensional, with three coordinates corresponding to initial co-states p_i and two corresponding to switch time and final time. Only two are represented. The diamond sets are constrained zonotopes, they enclose Monte Carlo solutions (orange) as expected.

As a side note, we also implemented a variant of the Inflate & Contract Algorithm that uses the resolvent-based propagation of boundary conditions of Section 4.1.2. We then compared the volume of the final box obtained with each constraint propagation technique and found that there was no noticeable difference, in fact the noise symbol variant yielded a box that was slightly smaller. However, the difference of computation time is of about a factor ten for our 1D version of Goddard's problem, and likely to be worse in higher dimension. Indeed, computing a resolvent requires simulating a system of dimension d^2 (or d systems of dimension $2d$ if one simulate the columns of the resolvent separately). As a consequence we focus mostly on the noise symbol propagation technique.

We now have a guaranteed enclosure of optimal co-states and switch times. We proceed to compute the anticipative enclosure. This is done by calling Algorithm 8 and having it export state enclosures $\mathbb{X}_n$ as well as constraints $\mathbb{A}$. Results are displayed In Figure 4.10. Optimal trajectories are enclosed as expected and optimality constraints compensate for the accumulation of errors caused by uncertainties.

In this chapter, we developed tools to overcome the difficulties of Chapter 3:

- we no longer use paving and are no longer limited by its exponential complexity,
- we no longer compute resolvents meaning the size of the dynamic system is proportional to d rather than d^2,
- symbolic zonotopes retain correlations between the initial co-state and subsequent states, and we no longer need to solve the OCP several times to compute the anticipative enclosure.

All of this significantly lowered the computational complexity of our method, so we now have methods that enclose the trajectory sets defined in Chapter 2 that are not limited to low dimension and can tackle hybrid behaviors. What remains to be seen is how well it fares on harder problems, and how those sets can be used in the context of guidance.

Chapter 5

Evaluating our methods on aerospace problems

The goal of this chapter is to assess to what extent our work can be applied to guidance of launchers. The goal is to first check whether our methods are capable of computing enclosures of launch vehicle trajectories, then list the information that can be deduced from these enclosures. To that end, we combine everything that was developed in previous chapters in a unified procedure, and confront it to aerospace optimal control problems. Then we investigate how those enclosures can be used.

5.1 Unifying the procedure

In order to assess the viability of our methods, we combine them in a unified procedure by which we tackle every Optimal Control Problem (OCP). It is summarized on Diagram 5.1, and we present it in greater detail below.

First, we transform our OCP into a hybrid Boundary Value Problem (BVP) (2.6) as described in Chapter 2 with pen and paper. We did not automate this task because the topic of optimal control is vast and there is no method that works on all problems, especially when state constraints are involved. Later sections will provide insight as to how many problem specific tweaks are required to tackle seemingly similar OCPs.

The OCP is inputted in our main algorithm as a sequence of dynamics $(g_n)_{n\in 0..N-1}$ and boundary constraint functions $(C_n)_{n\in 0..N-1}$, as well as jump functions $(j_n)_{n\in 0..N-1}$ when the OCP requires. These functions, as well as all our set based operations in our C++ code, are encoded with classes from the Ibex C++ Library [1]. Specifically Ibex-2.01 as it is the latest release that is compatible with our validated simulation library.

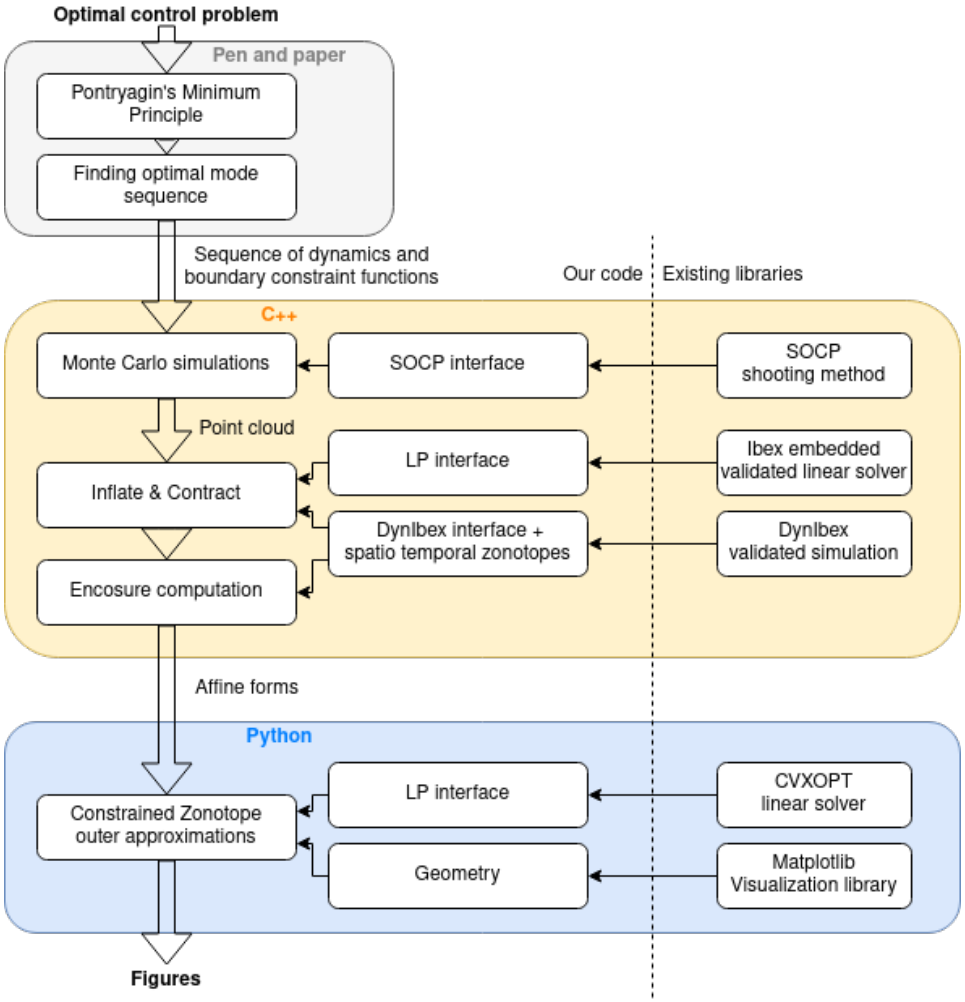


Figure 5.1: Summary of how we tackle optimal control problems.

Our algorithm starts with Monte Carlo simulations. For each simulation, we take one value amidst the set of possible parameters and possible initial states and solve the OCP with SOCP, a numerical shooting method developed by ONERA [20]. The values are chosen as follows: first we tackle every extreme case (if possible) then we complement with values taken uniformly at random until we have the desired sample size. For instance, if we have one initial state $y_i \in [y_i]$ and one parameter $\xi \in [\xi]$ and desire 100 sample points, we tackle the four extreme cases $(y_i = \underline{y_i}, \xi = \underline{\xi})$, $(y_i = \underline{y_i}, \xi = \overline{\xi})$, $(y_i = \overline{y_i}, \xi = \underline{\xi})$ and $(y_i = \overline{y_i}, \xi = \overline{\xi})$ then take 96 points where y_i takes a value in $[y_i]$ chosen uniformly at random and ξ does the same in $[\xi]$. In the event where the number $2^{d+d_{param}}$ of extreme cases exceeds the desired sample size, we only take random values. These Monte Carlo simulations give a cloud of initial co-states and switch times which will be used afterward, as well as a set of numerical trajectories to compare our sets to. Note that SOCP is numerical and not validated, hence we have no guarantee that those points and trajectories are solutions. Though since numerical errors are several orders of magnitude smaller than uncertainty induced over-approximation, we expect our sets to enclose those numerical solutions regardless.

We then compute our trajectory sets. The point cloud is used to initialize inflate & contract Method 2 with our Constrained Zonotope Contractor 9. We recall that this contractor requires validated simulation, which is done with DynIbex [4], a C++ library developed at ENSTA Paris that uses symbolic zonotopes. Constrained zonotopes and spatio-temporal zonotopes are handled by our interface rather than DynIbex itself, though results could be improved by hacking into DynIbex, as will be discussed in Section 5.2. The contractor uses Ibex embedded linear solver to compute the bounds of constrained zonotopes. This solver uses Neumaier Scherbina post-processing [50] to ensure all locally optimal co-states and switch times are enclosed. As a side note, we observed that this post-processing sometimes over-approximates the bounds by several orders of magnitude if there are generators that are too small, as the problem is numerically ill-posed. To avoid this, we do special reduction operations to cull all generators components below 10^{-12} and over-approximate them by a generator of norm at least 10^{-12}.

Then these initial co-states and switch times are used to compute an anticipative, open-loop or closed-loop enclosure. This is done by simulating respectively Dynamics (2.5), (2.9) or (2.10) with DynIbex and adding boundary constraints as zonotopes and exporting affine forms in text files.

Then we use a Python algorithm to compute outer-approximations of constrained zonotopes by bounding it in several directions using linear optimization (see Figure 1.5). Unlike our C++ solver, our Python solver is not validated, and we have not implemented Neumaier Scherbina post-processing for lack of time.

Making the python solver validated was not a priority because our Python algorithm is only used for visualization and numerical errors are likely smaller than the width of a line in our figures.

The results will only be compared to our Monte Carlo trajectories as there are no validated software that computes the same sets as us to the best of our knowledge. In any case our main concern is whether our procedure works at all since the inflate & contract is not guaranteed to converge. As a consequence, we challenge our method with increasingly complex problems.

5.1.1 Testing a range of aerospace problems

To evaluate our method, we run it on six test cases. Our main criterion is the size of uncertainties we can put in the model without causing our method to fail. Indeed, our procedure relies on an inflate & contract method that is not guaranteed to converge, but will converge if the uncertainties are small enough. The smaller the uncertainties, the more linear the BVP, the tighter our constrained zonotope contractor. For the purpose of the following study, we assume that all uncertainties are a percentage λ of the nominal value: $y_i \in [\hat{y}_i - \lambda\hat{y}_i, \hat{y}_i + \lambda\hat{y}_i]$ and $\xi \in [\hat{\xi} - \lambda\hat{\xi}, \hat{\xi} + \lambda\hat{\xi}]$. This is not realistic, but it unifies our test cases and allows for comparison: for each OCP we compute the biggest possible λ by trial and error. For all test cases, we compute 100 Monte Carlo trajectories and the final enclosure is always the anticipative enclosure.

Our test cases are built using the launcher model of Goddard's problem [19]:

$$\begin{aligned} &\min_{u(\cdot)} \int_{t_i}^{t_f} ||u|| dt, \\ &\text{s.t.} \begin{cases} \dot{r}(t) = v(t), & r(t_i) = r_i, \\ \dot{v}(t) = -\frac{G}{r^2} + \frac{C}{m}u - K\rho\frac{v}{||v||^3}, & v(t_i) = v_i, \\ \dot{m}(t) = -b||u(t)||, & m(t_i) = m_i. \end{cases} \end{aligned} \tag{5.1}$$

The state is composed of the position r, the velocity v and the mass m. The derivative of the position is velocity, the derivative of the mass is the sum of forces, which are gravity, thrust and drag force. For a take-off problem, $v(t_i) = 0$, and in 1D $r(t_i) = 1.00$, which correspond to a position on earth's crust because the model is normalized. The final velocity is free and there is a final constraint $r(t_f) = r_f = 1.01$ which sets the target. For a reentry problem it is the other way around: the initial state is in the air and initial speed is nonzero, while the final state is at the surface and the final velocity is null: $r(t_f) = 1.00$ and $v(t_f) = 0$. In all cases the initial mass starts at $m(t_i) = 1$ and the final mass is unconstrained. To sum up, in the take-off problem, only the initial mass m_i is subject to uncertainty while in the reentry problem position r_i, velocity v_i and mass m_i are subject to uncertainties.

ρ is the atmosphere density and will be either null, constant or a state variable depending on the test case. Parameters are composed of the normalized gravity field $G = 1$, the engine max thrust $C = 3.5$, the fuel consumption $b = 7$ and drag coefficient $K = 350$, their values are taken from [19]. We consider that these three parameters are uncertain but constant in each test case.

We built six versions of this problem of varying hardness. The most prominent driver of difficulty is dimension. Indeed, most of our test will be done in dimension 1, where the control and drag are binary: either upward or downward. In higher dimension, the control is a vector which considerably increases the system's nonlinearity.

Then second driver of difficulty is air drag, which also significantly increases nonlinearity. We take a gradual approach to air drag, starting with no atmosphere, then going to the case where the air density ρ is constant, then address the exponential model of atmosphere $\rho = \exp(-Ar)$.

Then there are state constraints. Although state constraints are a big hindrance when doing theory, in our case we consider simple dynamic pressure constraint, and they only change the control structure and add jumps.

Lastly, there is the matter whether it is a take-off or reentry problem. The former tend to be simpler because the initial position and velocity do not have uncertainty.

Our six test cases are sorted in general order of hardness in the following table:

#	Dimension	type	atmosphere	state constraint
1	1D	take-off	no	no
2	1D	take-off	no	$v \leq v_{max}$
3	1D	reentry	no	no
4	1D	reentry	$\rho = 1$	no
5	1D	reentry	$\rho = \exp(-Ar)$	no
6	2D	reentry	no	no

Test 1: take-off problem

We detail our procedure on this first test case. We start with a pen and paper analysis. The application of the PMP yields:

$$\begin{cases} H(y,p,u,\xi) = p_r v + p_v(-\frac{G}{r^2} + \frac{C}{m}u) - p_m b|u| \\ \dot{p_r}(t) = -2p_v(t)\frac{G}{r^3}, \\ \dot{p_v}(t) = -p_r(t), \\ \dot{p_m}(t) = p_v(t)\frac{C}{m^2}, \\ u = \begin{cases} \frac{p_v}{|p_v|} & \text{if } \Psi(t,y,p,\xi) > 0, \\ 0 & \text{if } \Psi(t,y,p,\xi) < 0, \end{cases} \end{cases} \tag{5.2}$$

where $\Psi(t,y,p,\xi) = C|p_v| - (1 + bp_m)m$.

We then deduce the optimal sequence of modes. Because fighting gravity costs fuel, the vehicle wants to escape it as fast as possible, hence the trajectory starts at full throttle: $u = 1$. Once the vehicle has enough momentum it turns its engine off to save fuel: $u = 0$. This translates as control:

$$u(t) = \begin{cases} 1 & \forall t \in [t_i, t_1], \\ 0 & \forall t \in [t_1, t_f]. \end{cases}$$

We write the resulting BVP using the notations of System (2.5):

$$\dot{x}(t) = g_n(x(t), \xi(t)), \forall t \in [t_{n-1}^+, t_n^-],$$
$$C_n(x(t_n^-)) = 0, \forall n \in 1, 2,$$
$$x = \begin{pmatrix} t \\ r \\ v \\ m \\ p_r \\ p_v \\ p_m \end{pmatrix}, g_1 = \begin{pmatrix} 1 \\ v \\ -\frac{G}{r^2} + \frac{C}{m} \\ -b \\ -2p_v\frac{G}{r^3} \\ -p_r \\ p_v\frac{C}{m^2} \end{pmatrix}, g_2 = \begin{pmatrix} 1 \\ v \\ -\frac{G}{r^2} \\ 0 \\ -2p_v\frac{G}{r^3} \\ -p_r \\ 0 \end{pmatrix},$$
$$C_1 = \begin{pmatrix} \Psi \end{pmatrix}, C_2 = \begin{pmatrix} r - r_f \\ p_v \\ p_m \\ H \end{pmatrix}.$$

We solve it for several values of the parameter and initial state, which yields the orange point cloud that initializes our inflate & contract method in Figure 5.2.

Then we compute the anticipative enclosure, and display them with our Python code, yielding Figure 5.3.

For the sake of example we assume that there is a velocity $v_{max} = 0.10$ that must not be exceeded. Since the unconstrained systems exceeds it, the OCP needs to add $v \leq v_{max}$ as a state constraint. The next section shows how to reformulate the BVP.

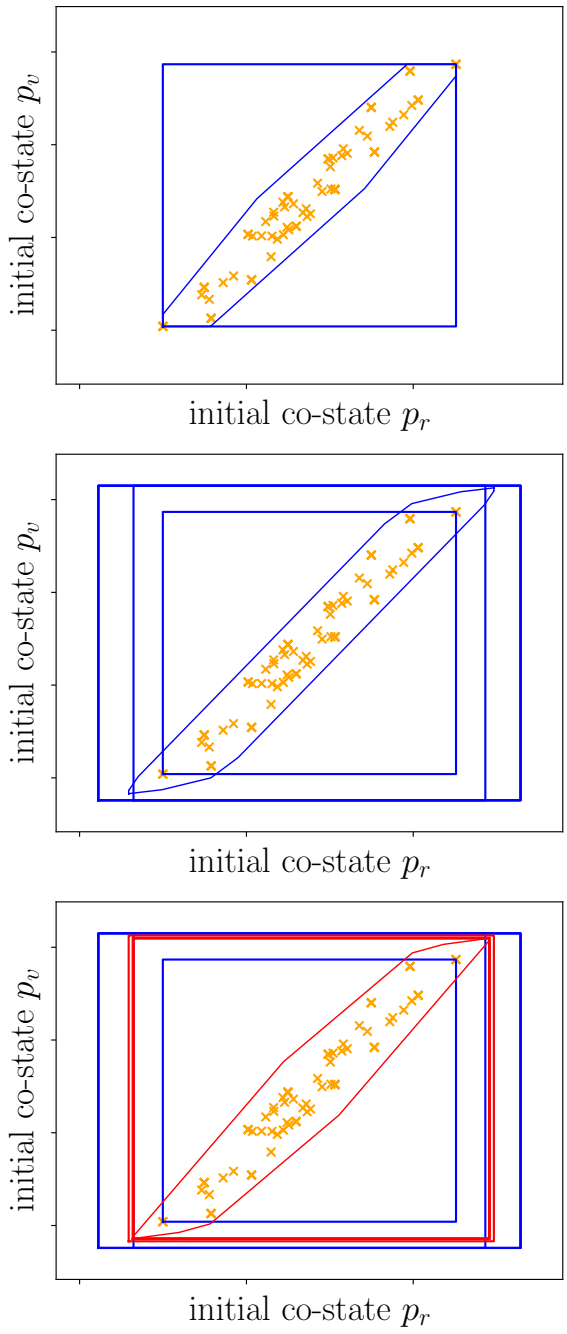


Figure 5.2: Inflate & Contract applied to Goddard's take-off problem, as in Figure 4.11 but with other coordinates. The final box is guaranteed to contain all local optimal co-states, as explained in Chapter 4.

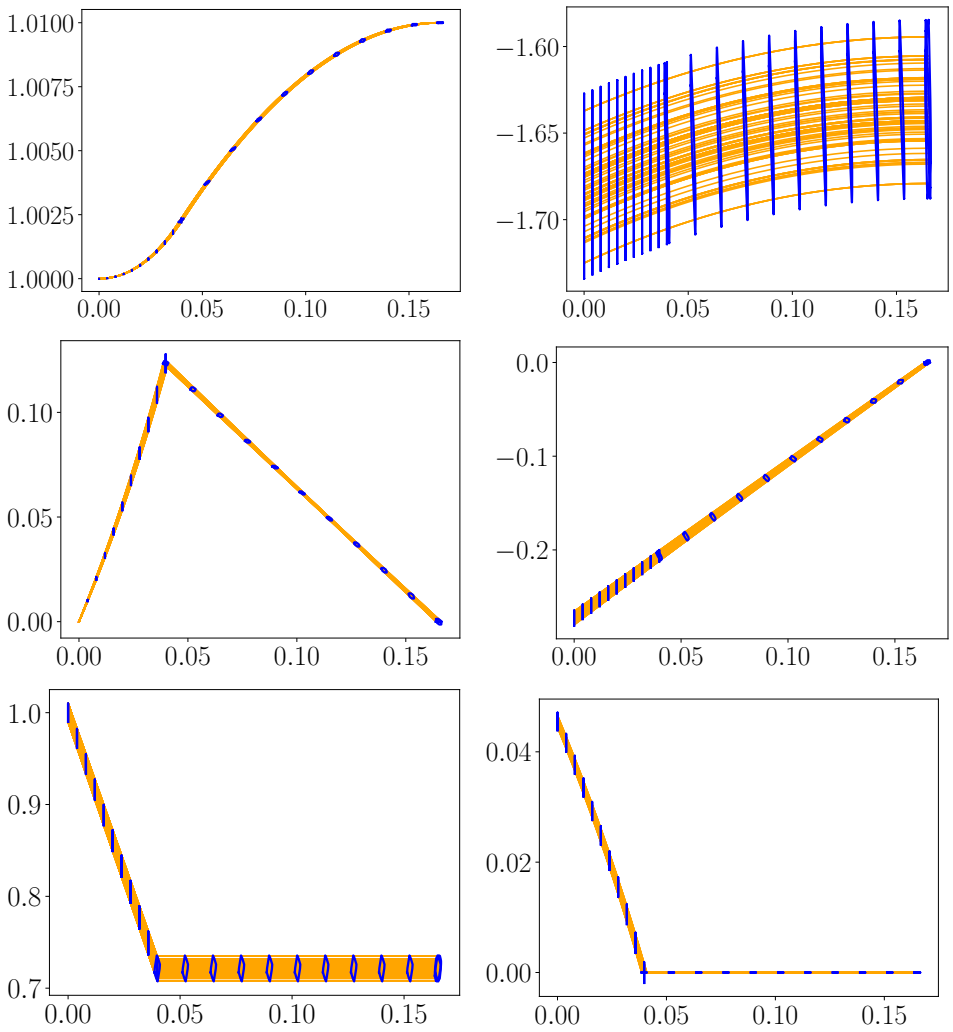


Figure 5.3: Coordinates of the extended state of Goddard's problem as a function of time. Left: Height, velocity, mass, right: co-state. The two modes of the trajectory are particularly visible on velocity and mass: during the full throttle phase, velocity increases and mass decreases while in free fall velocity decreases and mass stagnates. Our Monte Carlo trajectories (orange) are successfully enclosed by our constrained zonotopes (blue). Unlike in Figure 4.10, unconstrained zonotope hulls are omitted to improve readability.

Test 2: take-off with max velocity constraint

To avoid exceeding the maximum velocity, we ask the launcher to satisfy state constraint $v \leq v_{max}$. This changes the structure of the trajectory. Indeed, the vehicle reaches this maximum speed during the boost phase. At this point, the vehicle decreases its thrust just enough to not exceed the maximum velocity, which translates to $\dot{v} = 0 \implies u = Gm/Cr^2$. Then the trajectory ends in free fall meaning the control follows:

$$u(t) = \begin{cases} 1 & \forall t \in [t_i, t_1], \\ \frac{Gm}{Cr^2} & \forall t \in [t_1, t_2], \\ 0 & \forall t \in [t_2, t_f]. \end{cases}$$

However, there is also the possibility of a co-state jump when the constraint is reached. This jump writes $p_v(t_1^+) = p_v(t_1^-) + \mu_1$ and $p_v(t_2^+) = p_v(t_2^-) - \mu_2$, because the constraint is on v (see [38] for details).

This adds 2 degrees of freedom (μ_1 and μ_2), because there are always as many boundary constraints as variables, boundary constraints are:

$$C_1(t,y,p) = \begin{pmatrix} v - v_{max} \\ H(t,y,p) \end{pmatrix}, C_2(t,y,p) = \begin{pmatrix} C|p_v| - (1 + bp_m)m \\ H(t,y,p) \end{pmatrix}.$$

This uses the fact that the Hamiltonian is constant equal to 0, which is a consequence of the PMP when parameters are constant.

To sum up, the BVP is:

$$\begin{gathered} \dot{x}(t) = g_n(x(t), \xi(t)), \forall t \in [t_{n-1}^+, t_n^-], \\ p_v(t_1^+) = p_v(t_1^-) + \mu_1, p_v(t_2^+) = p_v(t_2^-) - \mu_2, \\ C_n(x(t_n^-)) = 0, \forall n \in 1, 2, 3, \end{gathered}$$

$$x = \begin{pmatrix} t \\ r \\ v \\ m \\ p_r \\ p_v \\ p_m \end{pmatrix}, g_1 = \begin{pmatrix} 1 \\ v \\ -\frac{G}{r^2} + \frac{C}{m} \\ -b \\ -2p_v\frac{G}{r^3} \\ -p_r \\ p_v\frac{C}{m^2} \end{pmatrix}, g_2 = \begin{pmatrix} 1 \\ v \\ 0 \\ -b\frac{Gm}{Cr^2} \\ -2p_v\frac{G}{r^3} \\ -p_r \\ p_v\frac{G}{mr^2} \end{pmatrix}, g_3 = \begin{pmatrix} 1 \\ v \\ -\frac{G}{r^2} \\ 0 \\ -2p_v\frac{G}{r^3} \\ -p_r \\ 0 \end{pmatrix},$$

$$C_1(x) = \begin{pmatrix} v - v_{max} \\ H \end{pmatrix}, C_2(x) = \begin{pmatrix} \Psi \\ H \end{pmatrix} C_3(x) = \begin{pmatrix} r - r_f \\ p_v \\ p_m \\ H \end{pmatrix}.$$

And its trajectories are enclosed in Figure 5.4.

The goal is to tackle reusable launchers, so after taking off, the system must return to its landing pad. Next section encloses reentry problems.

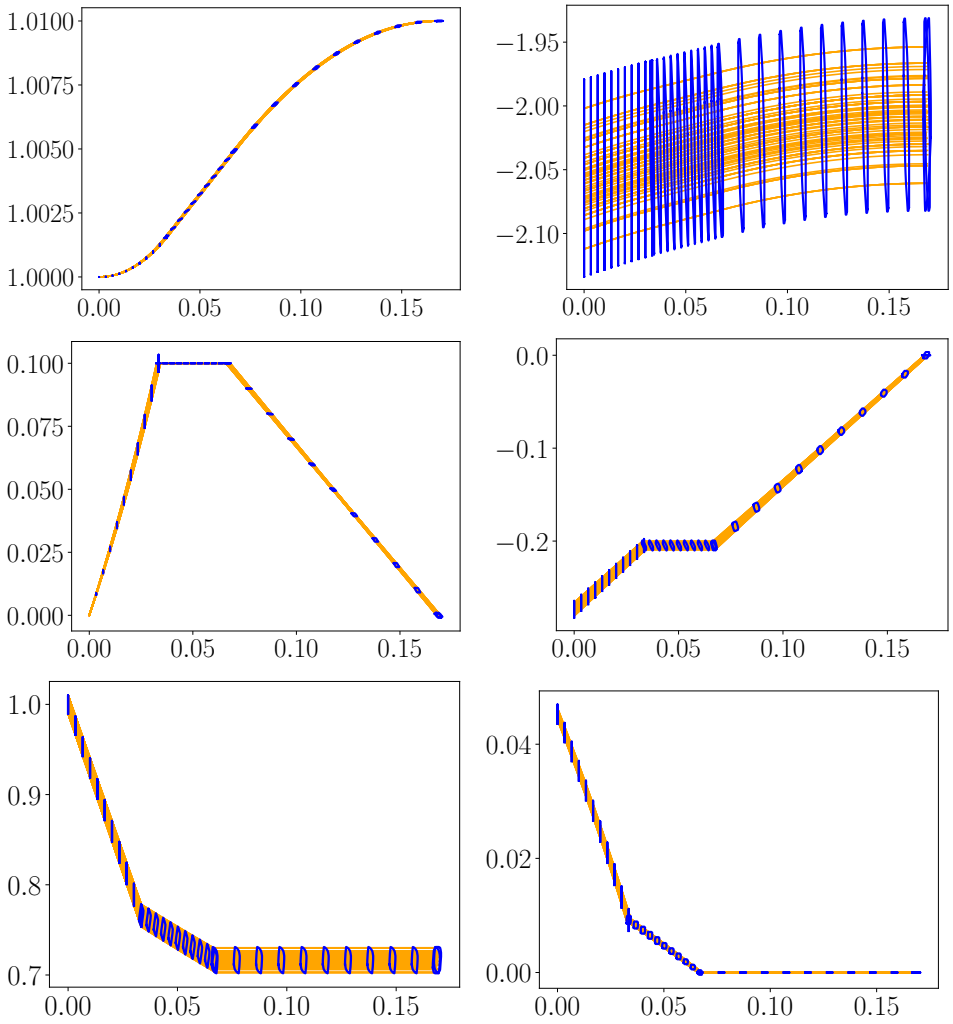


Figure 5.4: take-off with state constraint. The maximum velocity constraint $v \leq v_{max} = 0.1$ is visible on the velocity graph. Because those are mathematically perfect trajectories, they ride along the border of the unsafe set. As a consequence our zonotopes are horizontal (the vertical zonotope on the left of that trajectory segment belongs to the previous portion and is an over-approximation caused by spatio-temporal zonotopes). Although there could be a jump of co-state in theory, our numerical solver found none.

Tests 3-5: Reentry

A reentry problem starts in the air (typically at the end point of a take-off problem) and needs to finish on a landing site with a null velocity. The PMP takes the same form as with take-off problems, however the trajectory structure is different. Indeed, in 1D the system only needs to slow down at the end to reach the ground of a null velocity. As a consequence, the optimal control is:

$$\min_{u(\cdot)} \int_{t_i}^{t_f} |u| dt,$$
$$\text{s.t.} \begin{cases} \dot{r}(t) = v(t), & r(t_i) = r_i, \\ \dot{v}(t) = -\text{sign}(v) K\rho v^2 - \frac{G}{r^2} + \frac{C}{m}u, & v(t_i) = v_i, \\ \dot{m}(t) = -b|u(t)|, & m(t_i) = m_i, \\ u(t) = \begin{cases} 1, \forall t \in [t_i, t_1], \\ 0, \forall t \in [t_1, t_f]. \end{cases} \end{cases}$$

$$u(t) = \begin{cases} 0, \forall t \in [t_i, t_1], \\ 1, \forall t \in [t_1, t_2]. \end{cases}$$

Note that $sign(v)$ is problematic because it is not differentiable. In our test case, the initial velocity is downward, and it stays downward the entire trajectory. As a consequence we can assume that $sign(v) = -1$. In general, we would need to add additional modes for each case with a switch condition $v = 0$.

When ρ is constant, the BVP takes the form:

$$\dot{x}(t) = g_n(x(t), \xi(t)), \forall t \in [t_{n-1}^+, t_n^-],$$
$$C_n(x(t_n^-)) = 0, \forall n \in 1, 2,$$

$$x = \begin{pmatrix} t \\ r \\ v \\ m \\ p_r \\ p_v \\ p_m \end{pmatrix}, g_1 = \begin{pmatrix} 1 \\ v \\ K\rho v^2 - \frac{G}{r^2} \\ 0 \\ -2p_v \frac{G}{r^3} \\ -p_r \\ 0 \end{pmatrix}, g_2 = \begin{pmatrix} 1 \\ v \\ K\rho v^2 - \frac{G}{r^2} + \frac{C}{m} \\ -b \\ -2p_v \frac{G}{r^3} \\ -p_r \\ p_v \frac{C}{m^2} \end{pmatrix},$$

$$C_1 = \begin{pmatrix} \Psi \end{pmatrix}, C_2 = \begin{pmatrix} r - r_f \\ v \\ p_m \\ H \end{pmatrix}.$$

We first compute the anticipative enclosure with no atmosphere $\rho = 0$, the result is displayed in Figure 5.5. Then we compute with a constant atmosphere, the result is displayed in Figure 5.6.

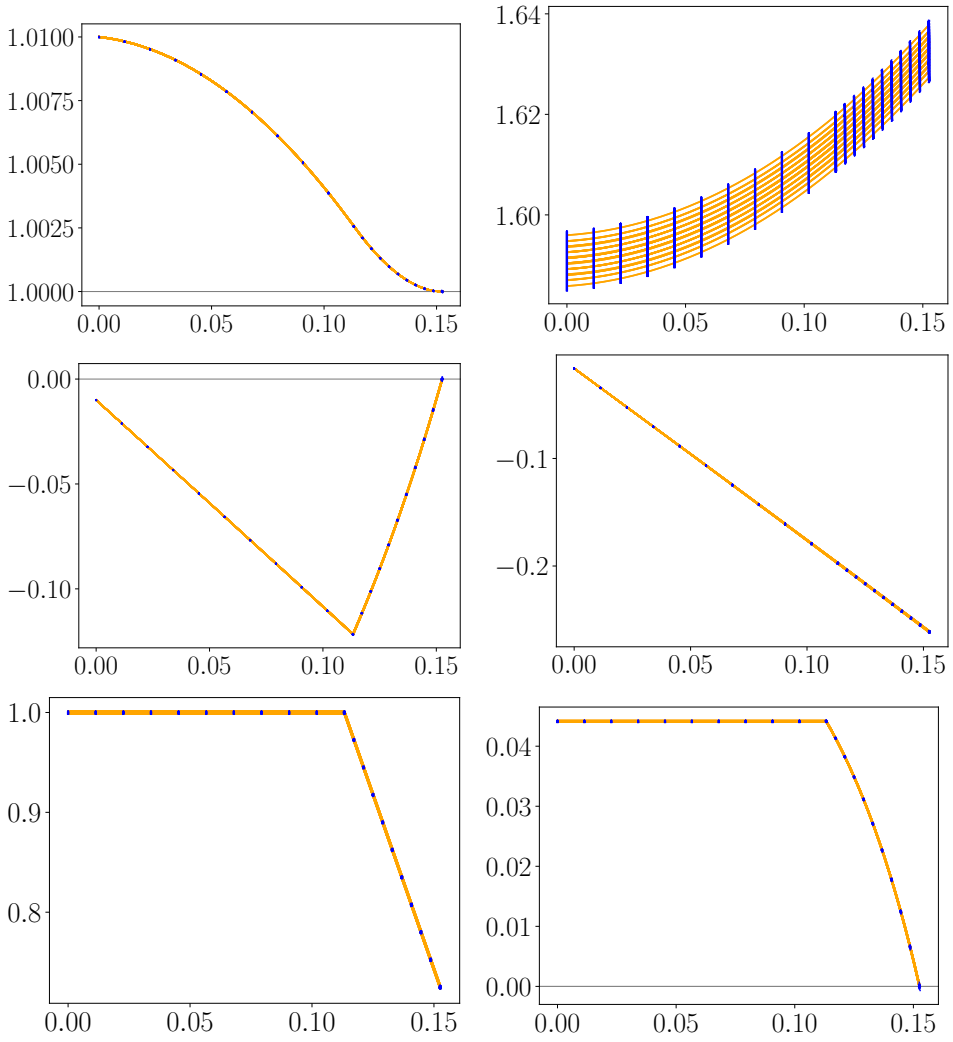


Figure 5.5: Reentry without drag force. It is like the take-off problem, but the other way around.

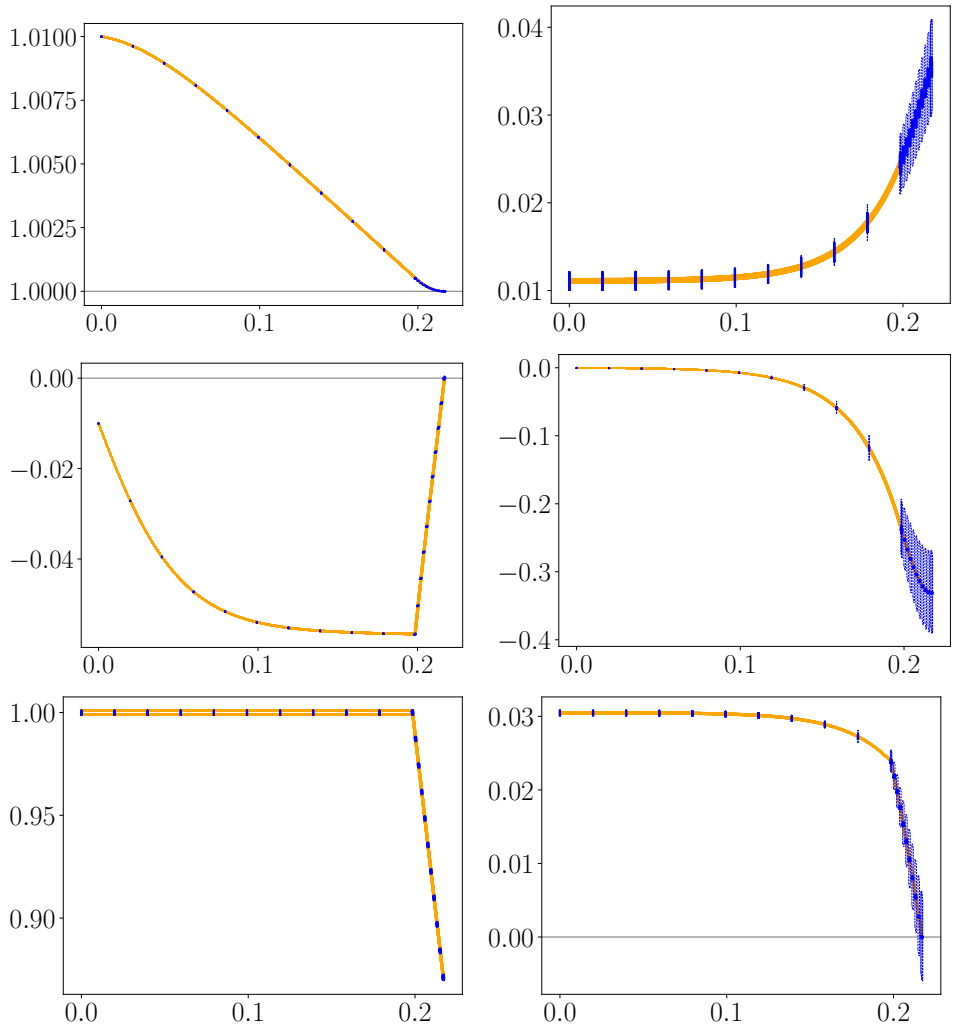


Figure 5.6: Reentry without constant atmosphere force. We can see the rocket reaching terminal velocity. The fact that co-state enclosures start very small but become very big at switch time is a sign that our spatio-temporal formula is imprecise.

Then we tried computing those enclosures with a more realistic model of the atmosphere $\rho = \exp(-A(r-1))$ The straightforward approach is to replace ρ by $\exp(-A(r-1))$ in OCP (5.1.1). However, the dynamics of the resulting BVP could not be enclosed by DynIbex. We are confident that this is because the exponential function is highly nonlinear and repeated calls caused the error to explode very fast. Nevertheless, we found a workaround by adding ρ as a coordinate to the state, yielding OCP (5.3) which we successfully enclosed, see Figure 5.7. We believe many problem specific adjustments of this caliber would be required to make our methods work on complex aerospace problems.

$$\min_{u(\cdot)} \int_{t_i}^{t_f} |u| dt,$$
$$\text{s.t.} \begin{cases} \dot{r}(t) = v(t), & r(t_i) = r_i, \\ \dot{v}(t) = -\text{sign}(v) K\rho v^2 - \frac{G}{r^2} + \frac{C}{m}u, & v(t_i) = v_i, \\ \dot{m}(t) = -b|u(t)|, & m(t_i) = m_i, \\ \dot{\rho}(t) = -A\rho v, & \rho(t_i) = \exp(-A(r_i - 1)), \\ u(t) = \begin{cases} 1, \forall t \in [t_i, t_1], \\ 0, \forall t \in [t_1, t_f]. \end{cases} \end{cases} \tag{5.3}$$

With that OCP, the BVP becomes:

$$\dot{x}(t) = g_n(x(t), \xi(t)), \forall t \in [t_{n-1}^+, t_n^-],$$
$$C_n(x(t_n^-)) = 0, \forall n \in 1, 2,$$
$$x = \begin{pmatrix} t \\ r \\ v \\ m \\ \rho \\ p_r \\ p_v \\ p_m \\ p_\rho \end{pmatrix}, g_1 = \begin{pmatrix} 1 \\ v \\ K\rho v^2 - \frac{G}{r^2} + \frac{C}{m} \\ -b \\ -A\rho v \\ -2p_v \frac{G}{r^3} \\ -p_r \\ p_v \frac{C}{m^2} \\ Kp_v v^2 + p_\rho A v \end{pmatrix}, g_2 = \begin{pmatrix} 1 \\ v \\ K\rho v^2 - \frac{G}{r^2} \\ 0 \\ -A\rho v \\ -2p_v \frac{G}{r^3} \\ -p_r + p_v A\rho \\ 0 \\ Kp_v v^2 + p_\rho A v \end{pmatrix},$$
$$C_1(x) = \begin{pmatrix} \Psi \end{pmatrix}, C_2(x) = \begin{pmatrix} r - r_f \\ v \\ p_m \\ p_\rho \\ H \end{pmatrix}.$$

It is enclosed in Figure 5.7.

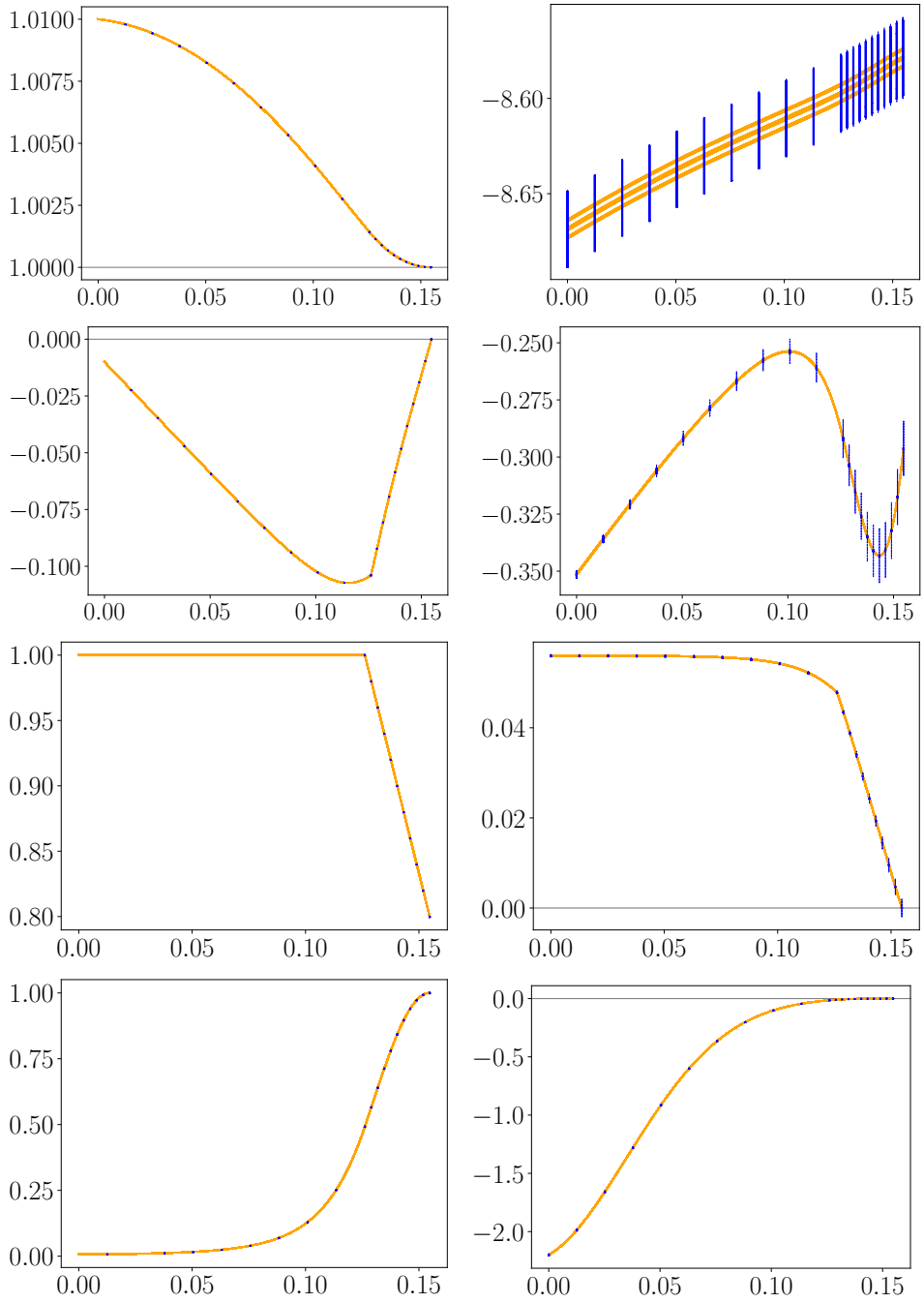


Figure 5.7: Optimal trajectories with realistic atmosphere. Last row is atmosphere density $\rho(t)$ and associated co-state. $\rho(t)$ starts close to 0 because the launcher is above the atmosphere, then increases exponentially as the launcher enters the atmosphere at high speed, but then drag force slows the launcher down. Hence, $\rho(t)$ is close to a logistic function.

2D dragless reentry

In this section, we increase the dimension of the state, velocity and associated co-states. This significantly complexifies the dynamics as the control is now a unitary vector. The initial state is the final state of the multi-dimensional take off-problem in [19], and the final state is its initial state. The goal is to emulate a reentry mission where the same platform serves as both launch and landing platform. In particular, the initial velocity vector of the reentry mission is in the wrong direction, since the previous take-off mission was getting away from the platform. So unlike previous reentry missions, the trajectory is in three phases and starts with a boost to turn around back toward the platform, then goes into free fall, then does a landing boost. This takes the form of the following BVP:

$$\dot{x}(t) = g_n(x(t), \xi(t)), \forall t \in [t_{n-1}^+, t_n^-],$$
$$C_n(x(t_n^-)) = 0, \forall n \in 1, 2,$$

$$x = \begin{pmatrix} t \\ r_1 \\ r_2 \\ v_1 \\ v_2 \\ m \\ p_{r1} \\ p_{r2} \\ p_{v1} \\ p_{v2} \\ p_m \end{pmatrix}, g_{1,3} = \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ -\frac{Gr_1}{||r||^3} + \frac{Cp_{v1}}{m||p_v||} \\ -\frac{Gr_2}{||r||^3} + \frac{Cp_{v2}}{m||p_v||} \\ -b \\ G\frac{p_{v1}||r||^2 - 3(p_v^T \cdot r)r_1}{||r||^5} \\ G\frac{p_{v2}||r||^2 - 3(p_v^T \cdot r)r_2}{||r||^5} \\ -p_{r1} \\ -p_{r2} \\ ||p_v||\frac{C}{m^2} \end{pmatrix}, g_2 = \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ -\frac{Gr_1}{||r||^3} \\ -\frac{Gr_2}{||r||^3} \\ 0 \\ G\frac{p_{v1}||r||^2 - 3(p_v^T \cdot r)r_1}{||r||^5} \\ G\frac{p_{v2}||r||^2 - 3(p_v^T \cdot r)r_2}{||r||^5} \\ -p_{r1} \\ -p_{r2} \\ 0 \end{pmatrix},$$

$$C_1 = \begin{pmatrix} \Psi \end{pmatrix}, C_2 = \begin{pmatrix} r_1 - r_{f1} \\ r_2 - r_{f2} \\ v_1 \\ v_2 \\ p_m \\ H \end{pmatrix}.$$

The results are displayed in Figure 5.8. While we do manage to enclose trajectories of this difficult problem, our blue constrained zonotopes in Figure 5.8 are so small they are barely visible. This is symptomatic of the main weakness of our approach, which is discussed in the next section.

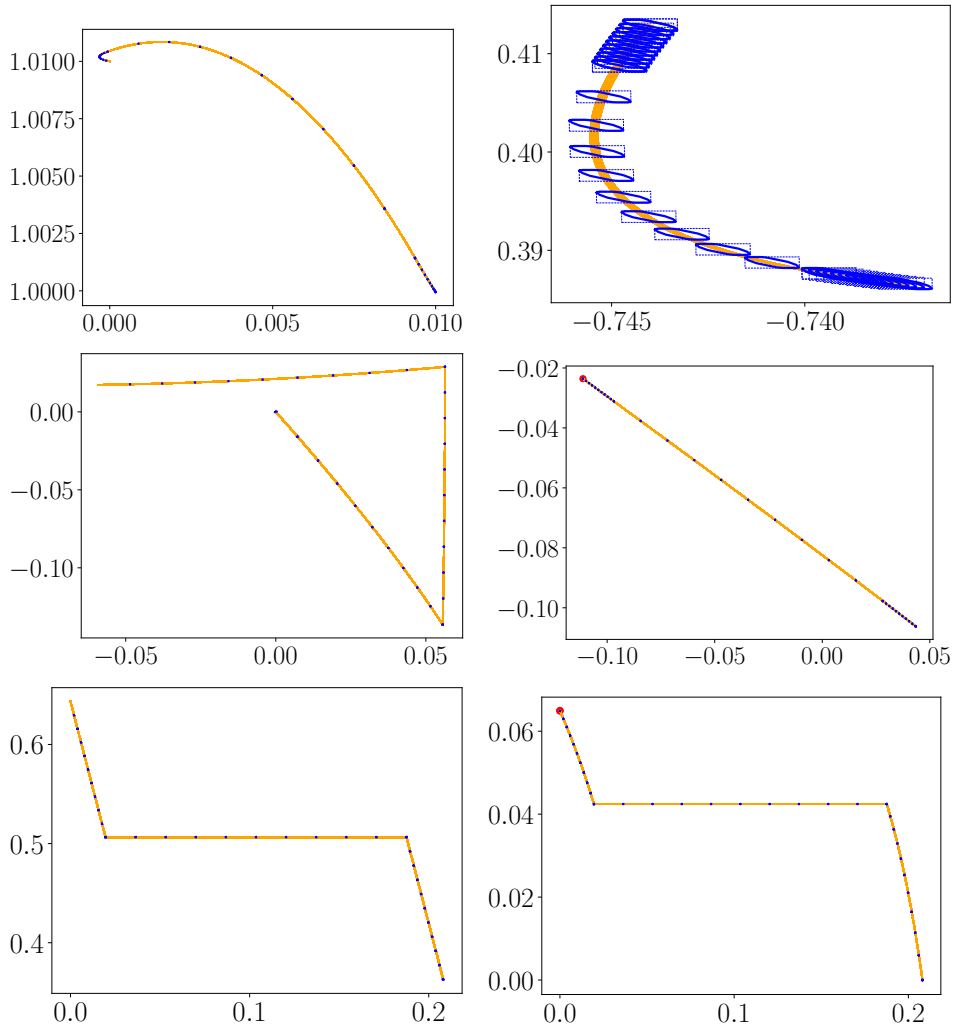


Figure 5.8: 2D reentry without drag force. Unlike previous systems, the upper four figures display trajectories in 2D rather than time functions. The vehicle starts at altitude 1.01 on the left, does a horizontal boost to turn around, then falls, then does an upward boost to reach the landing pad with a null velocity. This structure is especially visible on the velocity figure in the middle left: it starts on the left, then goes right to turn the launcher around, then falls due to gravity, then converges to 0.

5.1.2 Synthesis

As expected, the more difficult the problem, the least uncertainties we can put in our procedure because the inflate & contract fails. Table 5.1.2 synthesizes this phenomenon by listing the maximum magnitude of uncertainties we can put in a given test case. Those magnitudes were computed by trial and error: we run our code with 1% relative uncertainty on each parameter and uncertain initial state, then divide that magnitude by 10 when the inflate & contract algorithm fails to converge. Table 5.1.2 also lists computation time. All tests were done on a personal work computer with an AMD® Ryzen 5 4500u CPU. We add the integrator with quadratic cost as test case 0 for comparison.

#	Test case	uncertainty	run time	Infl iter	Contr
0	integrator	30%	70 seconds	11	24
1	take-off	3%	7 minutes	9	13
2	take-off, $v \leq v_{max}$	2%	25 minutes	11	31
3	reentry, $\rho = 0$	1%	7 minutes	30	40
4	reentry, $\rho = 1$	0.2%	1 hour	30	24
5	reentry, $\rho = \exp(-Ar)$	0.09%	2 hours	17	14
6	2D reentry	0.04%	10 hours	17	14

And computation time is spread as follows:

#	Monte Carlo	Infl&Contract	OuterEnclosure
0	1 minute	10 seconds	10 second
1	4 minutes	2 minutes	1 minute
2	11 minutes	9 minutes	4 minutes
3	4 minutes	2.5 minutes	45 seconds
4	15 minutes	45 minutes	2 minutes
5	1 hour	1 hour	2 minutes
6	5 hours	5 hours	10 minutes

"OuterEnclosure" corresponds to the Python code that solves LPs to display the enclosures. The 100 Monte Carlo simulations take a lot of time because each of them solves the OCP for different value of the parameters and initial state, so they each require a call to a nonlinear solver and several simulations. It is a lot more time-consuming than solving the OCP once, then doing 100 open-loop simulations around the nominal solution.

The next section draws conclusions on performance from this data.

5.2 Discussion

In this section, we make an overview of our methods to enclose trajectories: its performance, advantages and shortcomings.

5.2.1 Performance

Our procedure proved its ability to compute trajectory sets of moderately difficult OCPs in a couple of hours on an average CPU, and Monte Carlo simulations take a significant proportion of that time. This is due to the fact that our BVP solver does not require any discretization, hence its complexity does not scale exponentially with dimension like our initial paving-based algorithm. Our BVP solver is a single swipe of validated simulation, so the inflate & contract and enclosure computation as a whole is about tens of calls to validated simulation.

As a consequence, computation time mostly depends on how long it takes to simulate the system. Table 5.1.2 shows that the mere addition of drag force with constant atmosphere multiplies the inflate & contract time by 20, even though this does not increase the dimension of the system. That slowdown can be explained by the fact that DynIbex uses adaptive step size: when errors are too high as a result of the dynamics being complex, it decreases its simulation step until sufficiently small errors are achieved, and this increases the number of steps required to reach final time. Sadly, it is difficult to accelerate the program because the inflate & contract method is sequential, so it cannot be parallelized like paving or Monte Carlo methods.

The post treatment to display curves and compute enclosures can also take long, since displaying constrained zonotopes requires several calls to a linear solver, one for each direction, and those LPs have as many variables as there noise symbols in the constrained zonotope. However, this is highly customizable: the user can choose the number of directions they want for each enclosure, and they can also do a reduction operation on the zonotopes to decrease the size of the LPs. In our testing, the display LPs were not a significant proportion of the overall time.

In any case, computation time is not the biggest issue.

5.2.2 Failure to converge

The main weakness of our approach is that it relies on an inflate & contract method that is not guaranteed to converge. Our test case support our conjecture that the more nonlinear the system, the harder it is to converge, the smaller the uncertainties we can put in the model. While 1% relative uncertainty is good enough for some parameters and initial states like the mass at launch, it is insufficient to model air drag uncertainties that range in 10%.

The convergence rate, could be improved by improving the geometry of our method, we see three main routes:

- rotating the box of initial co-states and switch time to improve contraction,
- improving our spatio-temporal zonotope initialization with higher order Taylor or B-series [3],
- hacking in the validated simulation and affine arithmetic library so that it uses the constrained part of a constrained zonotope $\mathbb{Z}^{\mathbb{A}}$ rather than its unconstrained hull $\mathbb{Z}$, thus decreasing the size of the state set and decreasing over-approximations.

However, there is an inherent limitation in using zonotopes, as they only model linear correlations. Some problems will be too non-linear for them, and may require using other set representations, for instance polynomial constrained zonotopes as proposed in [43].

However, when the method does give results, those are validated.

5.2.3 Correctness

Our approach benefits from being correct by design: we have a proof that any optimal trajectory that fits the hypothesis of our model is in our enclosures. And that applies to a class of problems that are quite difficult, namely nonlinear hybrid systems which control is implicitly defined as minimizing an OCP. This is significant added value compared to probabilistic approaches like Monte Carlo, which never completely exclude the possibility of failure, or numerical methods that suffer from errors.

However, all our proofs are built on the assumption that our model with uncertainties is correct: that it encloses the actual dynamics of the vehicle. It is possible that a parameter or initial states exceeds the bounds of the model, as those are empirical.

As a side note, this is the reason why we put a lot of effort in generalizing our results and methods to the case where parameters are time functions rather than constant, even though DynIbex (and all the methods we built with it) can only simulate piecewise constant parameters. Indeed, in reality, some parameters (like atmosphere drag coefficients) are not constant, and many phenomena (like noises and disturbances) can only be modeled by a time dependent input. So if we started our reasoning with the assumption that parameters are constant, then all our proofs and results would have been considerably less applicable. Even though our code could not tackle variable parameters in the end, we hope to have convinced the reader that our methods could, given more information on the parameters, like Lipschitz bounds.

There is another model-related difficulty, in that our approach is complicated.

5.2.4 Need for preliminary work

As highlighted on Diagram 5.1, our procedure starts with a pen and paper analysis of the OCP, which requires advanced mathematical tools and knowledge of the optimal trajectory structure. Some difficulties subsist past the pen and paper step. We made a lot of effort to make our code open to strange trajectory structure, notably by encoding it as a sequence of dynamics and boundary constraints with little requirements on each. Yet we still had to hack in problem specific adjustments, notably the co-state jumps for OCPs with state constraints.

Moreover, our procedure may require additional adjustments to work, for instance our code could not enclose the test case with exponential atmosphere density until we added atmosphere density as a coordinate of the state.

Nevertheless, we envision potential use cases of these enclosures.

5.3 Perspectives

In this section, we propose ideas to use our enclosures.

5.3.1 Post-treatment of shooting method

Even when tackling problems with no uncertainties, numerical solvers are subject to rounding errors which will cause the result to differ from the actual solution. Our inflate & contract method and BVP solver could be used as post-processing to provide validated results. By enclosing the solution set, it can obtain safe bounds on the amount of error, hence prove that the optimal trajectory is within a certain distance of the numerical approximation given by the solver. Alternatively, new numerical trajectory and control could be computed by taking the centers of our enclosure state sets, and used as a reference trajectory.

There are however cases where a numerical solver converges to a solution that does not exist due to accumulated errors. One could check for this scenario by complementing the outer-approximation with an inner-approximation.

5.3.2 Inner-approximation method using symbolic zonotopes

Because our methods only compute outer-enclosures, they can detect the possibility of failure, but they cannot prove that there is indeed an actual scenario that leads to failure, unless the entire outer enclosure enters the unsafe set. There is always the possibility that an outer-enclosure crossed an unsafe set not because there are parameter values that lead there, but because of too much over-approximation. However, if one computes an inner-approximation, that is a set for which it is guaranteed that there exists an actual trajectory that goes through each point, and this set crosses the unsafe set, that proves the existence of a scenario where the system fails.

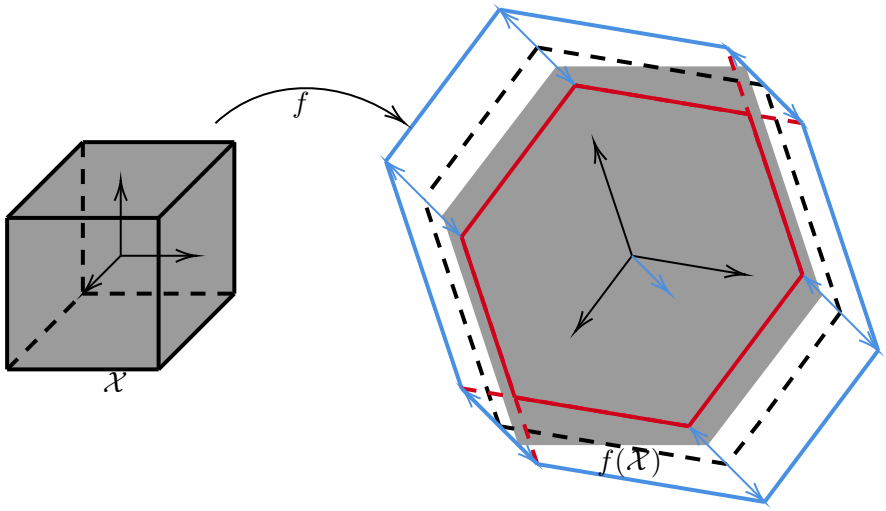


Figure 5.9: Computing an inner and outer-approximations at the same time. The gray set of inputs $\mathcal{X}$ on the left is enclosed in a zonotope which generators are the black arrows. Its image by a function f is approximated with affine arithmetic. The affine transformation of the black arrows form the black dashed zonotope which do not represent $f(\mathcal{X})$. Hence, the blue generator is added to enclose this error. Adding the blue arrow outward yields the blue outer-enclosure of $f(\mathcal{X})$, subtracting it inward yields the red inner-enclosure.

An inner approximation could be computed with post-treatment algorithm that flags noise symbols that correspond to coordinates of the input set $\mathcal{X}$ (typically those that correspond to initial state or parameter uncertainty) and noise symbols that correspond to over-approximations of errors. For a given zonotope $\mathbb{Z}$ that encloses the set $[f](\mathcal{X})$, we compute the Minkowski-sum of generators that correspond to noise symbol of $\mathcal{X}$, as for the classical outer-approximation, but we "subtract" generators that correspond to error noise symbols, as illustrated in Figure 5.9.

It can be proven with the fundamental invariant of affine arithmetic [29] that if f is continuous, then the inner set is subset of the true image $f(\mathcal{X})$. Thus, one could build an inner-approximations of the set of optimal trajectories, as in Figure 5.10.

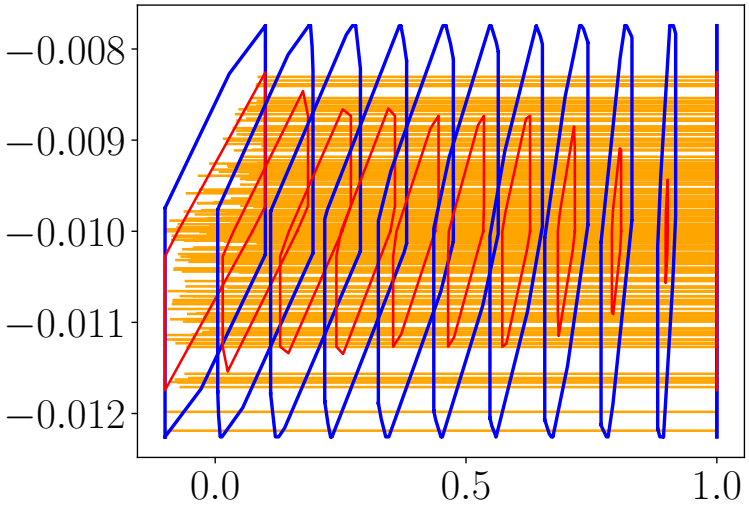


Figure 5.10: Inner and outer-approximation of the tube of optimal trajectories of an integrator with quadratic cost. Blue constrained zonotope enclose all trajectories, while each point of red inner sets are guaranteed to have at least one trajectory going through them.

Such inner and outer-enclosures can be used to assess robustness with respect to state constraints.

5.3.3 Assessing risks and computing margins

These enclosures can be used for reachability theory like the enclosures in [6]. The main application is to check if there is a chance to reach an unsafe set or violate a constraint of any kind.

If our outer enclosures are out of an unsafe set then that risk can be neglected. This is strong guarantee of robustness. Contrarily if the enclosure intersects with the unsafe set that can be enough to justify changing the controller of some critical system on which the possibility of failure is not tolerated. Our enclosure can help adapt a model.

Indeed, when faced with an unsafe set in optimal control, there are two main approaches. Either a state constraint is added to forbid the trajectory from going in the unsafe set, a cost penalization is added to desincentivize the system from going near the unsafe set. In both cases, the choice of the margin or penalization is crucial: if they are too small the system might enter the unsafe set, if they are too big it will impact performance.

Our enclosure can provide quantitative information on whether a given margin or penalization is enough, which can be used to find the right one as illustrated in Figure 5.11.

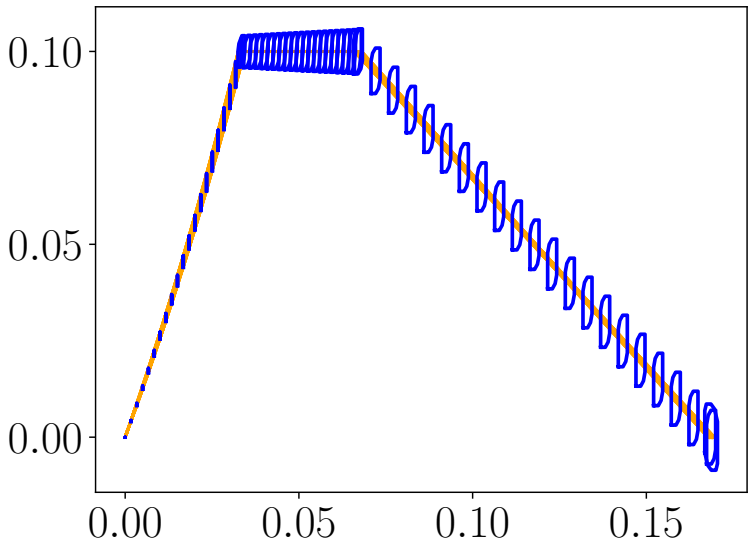


Figure 5.11: Open-loop enclosure of the velocity of take-off problem with state constraint $v \leq v_{max} = 0.10$. Unlike the mathematically perfect trajectories in Figure 5.4, trajectories in open-loop might exceed v_{max}. We could use the open loop enclosure to have an upper bounds on how high the velocity reaches for a given value of v_{max} and use this information to find an adequate safety margin to put below the actual unsafe velocity. This could be done through trial and error or a more advanced iterative algorithm.

Other information can be deduced by the computation of gradients.

5.3.4 Global Sensitivity analysis

Sensitivity analysis consists in computing how much a change of parameter value affects the trajectory as a whole. Local sensitivity analysis is done computing a resolvent, which gives the linear correlation between initial states or parameter values and subsequent states of the system. Global sensitivity analysis is generally done with statistical methods [39].

Our method could provide global guaranteed sensitivity analysis.

Firstly, validated simulation alone can generalize local sensitivity analysis by enclosing the resolvent of a dynamical system over a set of parameter values. This would give a linear end point mapping with error bounds. This can also be achieved cheaply simulating the system and using the correlation of the noise symbols.

Our methods are a little different in that they deal with OCPs, and trajectories are subject to a rebound effect that is not directly visible by simply computing a resolvent. For instance, if the thrust parameter decreases, then the burn will last longer to compensate, and we would not see this effect in the resolvent. However, this information is encapsulated in our constraint zonotope.

We can compute our enclosures for different values of the parameter (by adding a constraint $\xi = \hat{\xi}$ to our constrained zonotope) and thus approximate influence a given parameter has on the system.

Alternatively, we could compute the actual value of a parameter.

5.3.5 Parameter estimation

Calibration consists in estimating the true value of parameters with measurement. Interval-based validated calibration algorithms are proposed in [2] which consists in writing the correlation between parameters and noisy measurements as an interval constraint satisfaction problem, then enclosing its solutions.

We propose a similar algorithm using constrained zonotopes. Take a range $\mathbb{P}$ of parameters, compute an enclosure, add the measurement as constraints $\mathbb{A}$, then compute $\mathbb{P}^{\mathbb{A}}$. All values that are outside the outer enclosure $\mathbb{P}^{\mathbb{A}}$ are guaranteed to not be the actual values. Another upside is that we can apply it to optimal trajectories despite the previously mentioned rebound effect.

The idea to add measurement as constraint can also be used for navigation.

5.3.6 Online Guidance and Navigation

We put this case last because although control synthesis is the first thing that comes to mind, it is actually the most exotic use case of our algorithms because of the context of our work. Indeed, in the French literature, control of aerospace systems is composed of three blocks:

- trajectory generation: computing the reference trajectory, it is often done offline because it requires too much computing power for onboard electronics,
- guidance and navigation: correcting the trajectory using sensor data,
- attitude control: using actuators to follow the trajectory.

Control synthesis for trajectory generation corresponds to what we do in Sections 5.3.1, and 5.3.3: computing a trajectory and control in advance with safe bounds, margins and penalizations to ensure robustness.

Control synthesis for navigation on the other hand would rather consist in an online correction algorithm.

However, our algorithms are engineered for offline trajectory generation specifically, they cannot run online. Indeed, our algorithms are very expensive computationally (they take several hours on a computer) and they can fail to yield usable results (because of the inflate & contract method). But suppose that the system's dynamics are complicated enough that only a predictive controller can do the navigation, and its electronics cannot simulate the system online but can solve linear problems. Then Algorithm 10 could be used.

Algorithm 10: A controller based on our enclosure.

Data: Prior to the mission: an OCP, bounds on sensor errors.
During the mission: sensor data (y_k).

1 Prior to the mission
2 Compute the closed loop enclosure of the OCP.
3 $(\mathbb{Y}_k) \leftarrow$ observable quantities of the closed-loop enclosure, plus error bounds.
4 $(\mathbb{U}_k) \leftarrow$ controls of the closed-loop enclosure.
5 $\mathbb{A} \leftarrow$ optimality constraints.
6 During the mission
7 **while** *true* **do**
8 $\quad \mathbb{A} \leftarrow (\mathbb{A}; \mathbb{Y}_k - y_k)$.
9 $\quad u \leftarrow$ center of $\mathbb{U}_k^{\mathbb{A}}$.
10 $\quad$ use u as control.
11 $\quad k \leftarrow k + 1$.
12 **end**

Algorithm 10 is composed of two parts. Prior to the mission, it computes a zonotope tube with one state zonotope for each sampling time. Then it saves the observable quantities (states that can be later cross-referenced with sensor data) as well as the control zonotopes in the memory of the vehicle. During the mission, the navigation program adds sensor data as zonotope constraints (at Line 9), similarly to [59]. It then computes the center of control zonotope with optimality and sensor data constraints using a linear solver (at Line 9) and use it as control. The main benefit of this algorithm is that it does not need to simulate dynamical systems or solve nonlinear OCPs online. Instead, it only needs to keep zonotopes, hence matrices, in memory and solve LPs It also does parameter estimation online since zonotope constraints $\mathbb{A}$ from sensor data can be applied to all other zonotopes through the fundamental invariant of affine arithmetic, and in particular to the zonotope that encloses parameters.

The main weakness of this navigation algorithm is that $\mathbb{U}_k^{\mathbb{A}}$ could become empty, typically if the model and sensor error bounds that were used to compute the closed-loop enclosure did not correspond to the actual system.

There might be other ways to use our enclosures, but we have not explored them as this thesis comes to its conclusion.

Conclusion

Our goal was to combine indirect methods of optimal control with set-based methods to enclose sets of optimal trajectories for problems with uncertainties. To that end, we defined three sets of optimal trajectories, then studied the geometry of those sets and how they could be computed. We then put our theory to work by designing interval-based methods to enclose those sets. Those methods gave promising results but were limited to simple problems because they relied on expensive paving method to compensate for intervals' inherent wrapping effect. To tackle more complex problems, we developed techniques for zonotopes: spatio-temporal zonotopes to simulate hybrid systems and solve boundary value problem with ease. In addition to being contributions in their own right, those zonotope techniques allowed us to compute enclosures for a wider range of problems, even moderately complex aerospace problems. We streamlined our procedure and applied it to a range of increasingly complex problems. This allowed us to assess the main weakness of our approach: our method can fail to produce results, especially when the uncertainties are big and the system is highly nonlinear. Nevertheless, by decreasing the size of the uncertainty we obtained validated outer-approximations of our trajectory sets, and the computation time and errors are reasonable. We then proposed some use cases for these enclosures, like validated shooting method, robustness assessment, parameter adjusting and estimating and control synthesis for navigation.

In future works, we could improve the tightness of our zonotopes, or switch to other set representations, to tackle higher uncertainties and more complex problems. It could also be interesting to experiment on an actual vehicle. Lastly, we could develop our constrained zonotopes techniques further and apply them to other zonotope-based algorithms.

Bibliography

[1] IBEX : a C++ numerical library based on interval arithmetic and constraint programming. http://www.ibex-lib.org.

[2] J. Alexandre dit Sandretto. *Étalonnage des robots à câbles: identification et qualification*. PhD thesis, Université Nice Sophia Antipolis, 2013.

[3] J. Alexandre dit Sandretto. Set-based b-series. *Mathematics*, 10(17):3165, 2022.

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

[5] E. L. Allgower and K. Georg. *Introduction to numerical continuation methods*. SIAM, 2003.

[6] M. Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars*. PhD thesis, Technische Universität München, 2010.

[7] M. Althoff. An introduction to cora 2015. In *Proc. of the workshop on applied verification for continuous and hybrid systems*, pages 120–151, 2015.

[8] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2):233 – 249, 2010. IFAC World Congress 2008.

[9] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[10] Z. Artstein. Examples of stabilization with hybrid feedback. In *International Hybrid Systems Workshop*, pages 173–185. Springer, 1995.

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

[12] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[13] E. Bertin, E. Brendel, B. Hérissé, J. Alexandre dit Sandretto, and A. Chapoutot. Prospects on solving an optimal control problem with bounded uncertainties on parameters using interval arithmetics. *Acta Cybernetica*, Feb. 2021.

[14] E. Bertin, B. Hérissé, J. Alexandre dit Sandretto, and A. Chapoutot. Spatio-temporal constrained zonotopes for validation of optimal control problems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6708–6713. IEEE, 2021.

[15] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[16] L. Blackmore. Autonomous precision landing of space rockets. *The Bridge*, 4(46):15–20, 01 2016.

[17] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

[18] V. Boltyansky. Robust maximum principle in minimax control. *International Journal of Control*, 72(4):305–314, 1999.

[19] F. Bonnans, P. Martinon, and E. Trélat. Singular arcs in the generalized goddard's problem. *Journal of Optimization Theory and Applications*, 139(2):439–461, 2008.

[20] E. Brendel, B. Hérissé, and E. Bourgeois. Optimal guidance for Toss Back concepts of Reusable Launch Vehicles. In *EUCASS*, 2019.

[21] R. Bulirsch, J. Stoer, and J. Stoer. *Introduction to numerical analysis*, volume 3. Springer, 2002.

[22] J. C. Butcher. Coefficients for the study of runge-kutta integration processes. *Journal of the Australian Mathematical Society*, 3(2):185–201, 1963.

[23] J.-B. Caillau, M. Cerf, A. Sassi, E. Trélat, and H. Zidani. Solving chance constrained optimal control problems in aerospace via kernel density

estimation. *Optimal Control, Applications and Methods*, 39(5):1818–1832, 2018.

[24] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 183–192. IEEE, 2012.

[25] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[26] F. H. Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.

[27] F. H. Clarke and R. B. Vinter. The relationship between the maximum principle and dynamic programming. *SIAM Journal on Control and Optimization*, 25(5):1291–1311, 1987.

[28] C. Combastel and A. Zolghadri. A distributed kalman filter with symbolic zonotopes and unique symbols provider for robust state estimation in cps. *International Journal of Control*, 93(11):2596–2612, 2020.

[29] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, Rio de Janeiro, Brazil, 1997.

[30] A. V. Dmitruk and A. M. Kaganovich. The hybrid maximum principle is a consequence of pontryagin maximum principle. *Systems & Control Letters*, 57(11):964–970, 2008.

[31] J. M. Esposito, V. Kumar, and G. J. Pappas. Accurate event detection for simulating hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 204–217. Springer, 2001.

[32] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

[33] M. Garavello and B. Piccoli. Hybrid necessary principle. *SIAM Journal on Control and Optimization*, 43(5):1867–1887, 2005.

[34] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[35] A. Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[36] T. Haberkorn and E. Trélat. Convergence results for smooth regularizations of hybrid nonlinear optimal control problems. *SIAM Journal on Control and Optimization*, 49(4):1498–1522, 2011.

[37] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2008.

[38] R. F. Hartl, S. P. Sethi, and R. G. Vickson. A survey of the maximum principles for optimal control problems with state constraints. *SIAM review*, 37(2):181–218, 1995.

[39] B. Iooss and P. Lemaître. A review on global sensitivity analysis methods. *Uncertainty management in simulation-optimization of complex systems*, pages 101–122, 2015.

[40] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer London, 2001.

[41] R. Kalman. The theory of optimal control and the calculus of variations. *Mathematical optimization techniques*, 309:329, 1963.

[42] T. Kapela, M. Mrozek, P. Pilarczyk, D. Wilczak, and P. Zgliczynski. Capd-a rigorous toolbox for computer assisted proofs in dynamics. *Technical report*, 2010.

[43] N. Kochdumper and M. Althoff. Constrained polynomial zonotopes. *arXiv preprint arXiv:2005.08849*, 2020.

[44] E. Lindelöf. Sur l'application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 116(3):454–457, 1894.

[45] R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computerarithmetic*, pages 225–286, 1987.

[46] H. Maurer and J. Zowe. First and second-order necessary and sufficient optimality conditions for infinite-dimensional programming problems. *Mathematical programming*, 16(1):98–110, 1979.

[47] R. E. Moore. *Interval analysis*, volume 4. Prentice-Hall Englewood Cliffs, 1966.

[48] P. J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In *International Workshop on Hybrid Systems: Computation and Control*, pages 165–177. Springer, 1999.

[49] A. Neumaier and A. Neumaier. *Interval methods for systems of equations*. Number 37. Cambridge university press, 1990.

[50] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2):283–296, 2004.

[51] É. Picard. Sur l'application des méthodes d'approximations successives à l'étude de certaines équations différentielles ordinaires. *Journal de mathématiques pures et appliquées*, 9:217–272, 1893.

[52] B. Piccoli. Hybrid systems and optimal control. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, volume 1, pages 13–18. IEEE, 1998.

[53] L. S. Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[54] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[55] H. Rabitz, M. Kramer, and D. Dacol. Sensitivity analysis in chemical kinetics. *Annual review of physical chemistry*, 34(1):419–461, 1983.

[56] N. Ramdani and N. S. Nedialkov. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162, 2011. Special Issue related to IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09).

[57] A. Rauh and E. P. Hofer. Interval methods for optimal control. In G. Buttazzo and A. Frediani, editors, *Variational Analysis and Aerospace Engineering*, chapter 22, pages 397–418. Springer New York, New York, NY, 2009.

[58] A. Rauh, J. Minisini, and E. P. Hofer. Interval techniques for design of optimal and robust control strategies. In *12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006), Duisburg,*, 2006.

[59] B. S. Rego, G. V. Raffo, J. K. Scott, and D. M. Raimondo. Guaranteed methods based on constrained zonotopes for set-valued state estimation of nonlinear discrete-time systems. *Automatica*, 111:108614, 2020.

[60] J. Richalet, A. Rault, J. Testud, and J. Papon. Model predictive heuristic control. *Automatica (journal of IFAC)*, 14(5):413–428, 1978.

[61] S. Romig, L. Jaulin, and A. Rauh. Using interval analysis to compute the invariant set of a nonlinear closed-loop control system. *Algorithms*, 12(12):262, 2019.

[62] M. G. Safonov. Origins of robust control: Early history and future speculations. *Annual Reviews in Control*, 36(2):173–181, 2012.

[63] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126 – 136, 2016.

[64] E. Trélat. *Contrôle optimal : théorie et applications*. Vuibert, 2005.

[65] E. Trélat. Optimal Control and Applications to Aerospace: Some Results and Challenges. *JOTA*, 154(3), 2012.

[66] R. B. Vinter. Minimax optimal control. *SIAM journal on control and optimization*, 44(3):939–968, 2005.

[67] R. B. Vinter and R. Vinter. *Optimal control*. Springer, 2010.

ECOLE
DOCTORALE

Titre : Contrôle Optimal et Robuste pour le Guidage de Véhicules Autonomes

Mots clés : Méthodes Ensemblistes, Simulation validée, Intervalles, Zonotopes, Contrôle Optimal, Principe du Maximum Pontryagin

Résumé : Le guidage d'un lanceur réutilisable est un problème de contrôle qui nécessite à la fois précision et robustesse : il faut calculer une trajectoire et un contrôle, de sorte que le lanceur atteigne la piste d'atterrissage, sans s'écraser ni exploser en vol, le tout en utilisant le moins de carburant possible. Les méthodes de Contrôle Optimal issu du Principe de Pontryagin calculent une trajectoire optimale avec grande précision, mais les incertitudes, soit les erreurs entre les estimations de l'état initial et des paramètres et leurs valeurs réelles, causent une déviation potentiellement dangereuse de la trajectoire réelle. En parallèle, les méthodes ensemblistes et notamment la simulation validée peuvent encadrer toutes les trajectoires possibles d'un système dynamique avec des incertitudes bornées. Cette thèse combine ces deux approches pour encadrer des ensembles de trajectoires optimales de systèmes avec incertitudes afin de garantir la robustesse du guidage d'un véhicule autonome.

Nous commençons par définir des ensembles de trajectoires optimales pour des systèmes avec incertitudes, d'abord pour les trajectoires mathématiquement parfaites, puis pour les trajectoires d'un véhicule sujet à des erreurs d'estimation, mais qui utiliserait, ou non, les données des capteurs pour recalculer sa trajectoire en cours de route. Le principe de Pontryagin caractérise ces ensembles comme solutions de problèmes aux deux bouts avec des dynamiques avec incertitudes. Nous développons alors des algorithmes qui encadrent toutes les solutions de ces problèmes aux deux bouts en utilisant la simulation validée, l'arithmétique des intervalles et la théorie des contracteurs. Cependant, la simulation avec des intervalles occasionne une forte sur-approximation qui limite nos méthodes. Pour y remédier, nous remplaçons les intervalles par des zonotopes symboliques contraints. Nous utilisons notamment ces zonotopes pour simuler des systèmes hybrides, encadrer des solutions de problèmes aux deux bouts et construire des sous-approximations en complément de la sur-approximation classique. Enfin, nous combinons tout ceci pour calculer des ensembles de trajectoires de systèmes aérospatiaux et les utilisons pour évaluer la robustesse du contrôle.

Title : Robust Optimal Control for Guidance of Autonomous Vehicles

Keywords : Set-Based Methods, Validated Simulation, Intervals, Zonotopes, Optimal Control, Pontryagin's Maximum Principle

Abstract : The guidance of a reusable launcher is a control problem that requires both precision and robustness: one must compute a trajectory and a control such that the system reaches the landing zone, without crashing into it or exploding mid-flight, all while using as little fuel as possible. Optimal control methods based on Pontryagin's Maximum Principle can compute an optimal trajectory with great precision, but uncertainties, the discrepancies between estimated values of the initial state and parameters and actual values, cause the actual trajectory to deviate, which can be dangerous. In parallel, set-based methods and notably validated simulation can enclose all trajectories of a system with uncertainties. This thesis combines those two approaches to enclose sets of optimal trajectories of a problem with uncertainties to guarantee the robustness of the guidance of autonomous vehicles.

We start by defining sets of optimal trajectories for systems with uncertainties, first for mathematically perfect trajectories, then for the trajectory of a vehicle subject to estimation errors that can use, or not use, sensor information to compute a new trajectory online. Pontryagin's principle characterizes those sets as solutions of a boundary value problem with dynamics subject to uncertainties. We develop algorithms that enclose all solutions of these boundary value problem using validated simulation, interval arithmetic and contractor theory. However, validated simulation with intervals is subject to significant overapproximation that limits our methods. To remedy that we replace intervals by constrained symbolic zonotopes. We use those zonotopes to simulate hybrid systems, enclose the solutions of boundary value problems and build an inner-approximation to complement the classical outer-approximation. Finally, we combine all our methods to compute sets of trajectories for aerospace systems and use those sets to assess the robustness of a control.

Institut Polytechnique de Paris
91120 Palaiseau, France