

# PHD THESIS

ECOLE NATIONALE SUPERIEURE  
DE TECHNIQUES AVANCÉES BRETAGNE  
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601  
*Mathématiques et Sciences et Technologies  
de l'Information et de la Communication*  
Branch : Robotics

Presented by

**Thomas LE MÉZO**

## **Bracketing largest invariant sets of dynamical systems**

An application to drifting underwater robots in ocean currents

Thesis defended at Brest on December 5, 2019  
Laboratory : Lab-STICC

### **Rapporteurs :**

David Daney Senior Researcher, Inria, Bordeaux  
Antoine Girard Senior Researcher CNRS, L2S, Gif-sur-Yvette

### **Jury composition :**

President :	Thao Dang	Research Director CNRS, Verimag, Saint Martin d'Herès
Reviewer :	Reda Boukezzoula Nicolas Delanoue	Professor, LISTIC, Annecy-le-vieux Associate Professor, LARIS, Angers
Thesis co-directors :	Luc Jaulin Benoit Zerr	Professor, Lab-STICC, Brest Professor, Lab-STICC, Brest
Thesis co-supervisor :	Damien Massé	Associate Professor, Lab-STICC, Brest



# Bracketing largest invariant sets of dynamical systems

An application to drifting underwater robots in ocean currents

Thomas Le Mézo

January 10, 2020

## Acknowledgments

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Motivation and background . . . . .	6
1.1.1	Using robots in the ocean . . . . .	6
1.1.2	The problem of drifting underwater robots . . . . .	9
1.1.3	Safety of robotic systems . . . . .	11
1.2	Contributions and Outlines . . . . .	12
1.3	Softwares . . . . .	13
<b>2</b>	<b>Tools to handle dynamical systems</b>	<b>15</b>
2.1	Dynamical systems . . . . .	16
2.1.1	Definition . . . . .	17
2.1.2	Flow map and properties of dynamical systems . . . . .	18
2.1.3	Set and lattice . . . . .	23
2.1.4	Invariant sets . . . . .	25
2.1.5	Extension of the dynamical system model . . . . .	28
2.1.6	Lyapunov theory . . . . .	32
2.2	Abstract domains . . . . .	34
2.2.1	Abstract Interpretation . . . . .	36
2.2.1.1	Definitions . . . . .	37
2.2.1.2	Properties of abstract domains . . . . .	41
2.2.2	Example of abstract domains . . . . .	43
2.2.2.1	Intervals . . . . .	44
2.2.2.2	Convex Polytopes . . . . .	50
2.2.2.3	Paving and subpaving . . . . .	53
2.2.3	The choice of an abstract domain . . . . .	56
2.3	Constraint programming . . . . .	57
2.3.1	Constraint network and fixed points . . . . .	57
2.3.1.1	Definition . . . . .	57
2.3.1.2	Fixed point . . . . .	58
2.3.2	Bracketing the solution set of a Constraint Network . . . . .	62
2.3.2.1	Outer approximation . . . . .	63

2.3.2.2	Inner approximation . . . . .	64
2.3.2.3	Example . . . . .	65
2.3.3	Example of algorithms . . . . .	68
2.3.3.1	CSP and Set inversion problem . . . . .	69
2.3.3.2	Dealing with datasets . . . . .	71
2.4	Conclusion . . . . .	77
<b>3</b>	<b>Mazes: a new abstract domain for paths</b>	<b>79</b>
3.1	Definitions and problem statement . . . . .	82
3.2	Mazes . . . . .	83
3.2.1	Roads . . . . .	84
3.2.2	Inclusion and lattice . . . . .	85
3.2.3	Door consistency . . . . .	87
3.3	Method and algorithm . . . . .	88
3.3.1	Computing an outer approximation for $\text{Inv}_f^+(\mathbb{X})$ . . . . .	89
3.3.2	Computing an inner approximation for $\text{Inv}_f^+(\mathbb{X})$ . . . . .	91
3.3.3	Main Algorithm . . . . .	97
3.3.3.1	Paving bisection . . . . .	97
3.3.3.2	An algorithm to bracket the largest positive invariant set . . . . .	99
3.3.3.3	Computed enclosure . . . . .	101
3.3.3.4	Complexity . . . . .	101
3.3.4	The Van Der Pol example . . . . .	103
3.3.5	Parameters that affect the speed of convergence . . . . .	103
3.4	Toward an implementation . . . . .	108
3.4.1	The sliding issue . . . . .	108
3.4.1.1	Problem statement . . . . .	108
3.4.1.2	Graph model of Mazes . . . . .	109
3.4.1.3	Graph rebuilding . . . . .	111
3.4.2	Using abstract domains without the ACC . . . . .	114
3.4.3	Constraint propagation heuristic and multithreading capabilities . . . . .	114
3.4.4	Using existing libraries . . . . .	116
3.5	Differential inclusion & system with input . . . . .	119
3.5.1	Outer approximation . . . . .	119
3.5.2	Inner approximation . . . . .	120
3.5.3	The example of the reverse Van Der Pol system with an input . . . . .	122
3.6	Conclusion . . . . .	123

<b>4</b>	<b>Applications of invariant sets</b>	<b>125</b>
4.1	The largest positive and negative invariant sets . . . . .	126
4.1.1	The problem . . . . .	126
4.1.2	Application: the example of isobath navigation . . . . .	127
4.2	Forward & Backward reach sets . . . . .	134
4.3	Attraction basin . . . . .	141
4.4	Capture reach set . . . . .	145
4.5	Eulerian state estimation . . . . .	147
4.5.1	Formalism . . . . .	149
4.5.2	Invariant sets approach . . . . .	149
4.6	Conclusion . . . . .	156
<b>5</b>	<b>A hybrid profiling float</b>	<b>157</b>
5.1	Problem formalization . . . . .	158
5.1.1	The mission . . . . .	158
5.1.1.1	Use case . . . . .	158
5.1.1.2	Mission formalization . . . . .	158
5.1.2	The robot . . . . .	161
5.2	Float dynamics . . . . .	162
5.3	Design of a hybrid profiling float . . . . .	165
5.3.1	Mechanical architecture . . . . .	165
5.3.1.1	Float hull . . . . .	166
5.3.1.2	Float compressibility . . . . .	168
5.3.1.3	Auto-ballasting system . . . . .	168
5.3.1.4	Additional systems . . . . .	169
5.3.2	Electronic architecture . . . . .	170
5.3.3	Software architecture . . . . .	171
5.4	Depth controller . . . . .	171
5.4.1	Control law . . . . .	172
5.4.2	Estimation of unknown parameters . . . . .	174
5.4.3	Experimental results . . . . .	175
5.5	Validation of the depth control law . . . . .	181
5.5.1	Open-loop float performances . . . . .	181
5.5.2	Closed-loop system . . . . .	185
5.5.3	Additional validations . . . . .	187
5.5.3.1	Vector field following . . . . .	187
5.5.3.2	Minimum piston volume increment . . . . .	188
5.5.3.3	Energy consumption . . . . .	188
5.6	Design loop . . . . .	189
5.7	Conclusion and future work . . . . .	190

<b>6</b>	<b>General conclusion and prospects</b>	<b>191</b>
6.1	Summary of the Contributions . . . . .	191
6.2	Prospects . . . . .	192



# Chapter 1

## Introduction

### Contents

---

<b>1.1</b>	<b>Motivation and background</b>	<b>6</b>
1.1.1	Using robots in the ocean	6
1.1.2	The problem of drifting underwater robots	9
1.1.3	Safety of robotic systems	11
<b>1.2</b>	<b>Contributions and Outlines</b>	<b>12</b>
<b>1.3</b>	<b>Softwares</b>	<b>13</b>

---

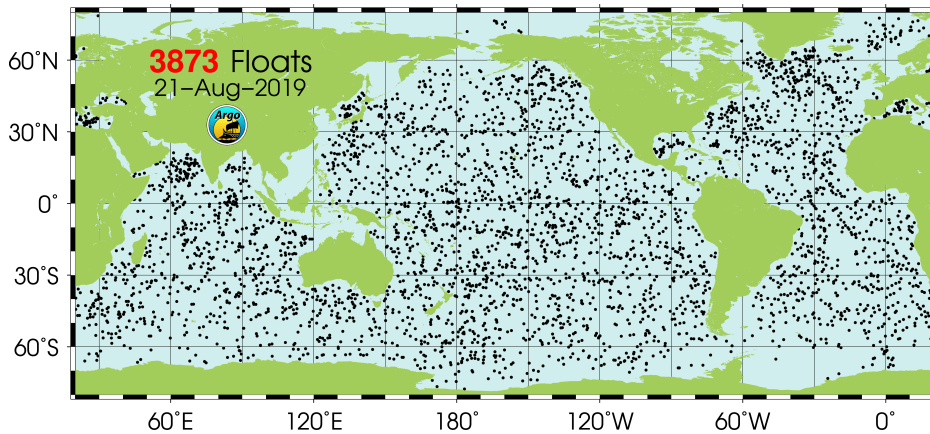


Figure 1.1: Distribution of profiling floats used in the Argo project that gather biochemical data in the ocean (<http://www.argo.ucsd.edu/>).

## 1.1 Motivation and background

### 1.1.1 Using robots in the ocean

The number of deployed robots in the oceans has increased substantially the last past decade. Indeed, they enable long-duration missions in harsh environments to be carried out. Figure 1.1 shows, for instance, the distribution of the nearly 4000 robots of the Argo project in August 2019 that are profiling every day, autonomously, the oceans to gather biochemical data. They are the sensors of a wide program that aims to better understand and predict the evolution of oceans.

Nowadays, marine robots are used to achieve an increasing number of tasks with a varying degree of autonomy from human operated to completely autonomous missions. Several kinds of marine robots can be distinguished (Creuze, 2014): Figure 1.2 shows a non exhaustive taxonomy of the main categories. We can firstly identify Unmanned Surface Vehicles (USV), that are mostly autonomous boats, and Unmanned Underwater Vehicles (UUV), that are mainly submarine robots.

In this work, we will focus on UUV and more specifically on Autonomous Underwater Vehicles (AUV). This kind of robots has the particularity of not being linked to the surface with a wire unlike Remotely Operated Vehicles (ROV). Therefore, they need to be largely autonomous because communications are difficult underwater. Several kinds of AUVs can be also distinguished: torpedo robots, gliders and profiling floats. Torpedo robots are usually equipped with propellers and fins: they are the fastest AUVs. Gliders use underwater wings and a variable buoyancy to travel, at small

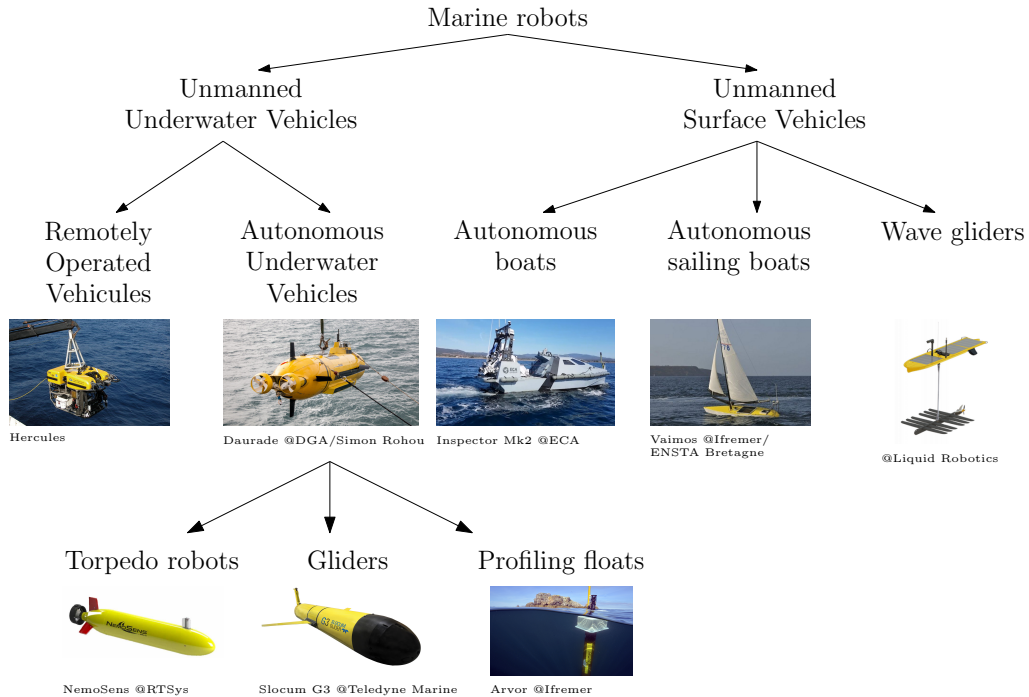


Figure 1.2: Marine robot taxonomy.

velocities, through long distances without any thrusters. Profiling floats can only evolve vertically and are dependent of the ocean currents for their horizontal movements. They also do not have any thrusters.

Several users of underwater robots can be identified, with specific missions (Zereik et al., 2018).

- The *Scientific community* uses marine robots mainly to monitor the ocean. For instance, gliders and profiling floats gather data such as the temperature, the salinity, the pH, the chlorophyll concentration, etc. The Argo project (Riser et al., 2016) is one of the most emblematic project that uses several thousands of profiling floats to measure the oceans. There is also some applications with water archeology.
- The *Defense and State* uses mainly AUVs with applications such as bathymetry (mapping of the sea ground, see Figure 1.3), search of wrecks and also mine-hunting. Some autonomous boats are used for surveillance missions.
- The *Industry* deploys since the 1980s an important number of ROVs in the oil and gas sector to build, maintain and repair offshore infrastructures. Several new applications that use marine robots have

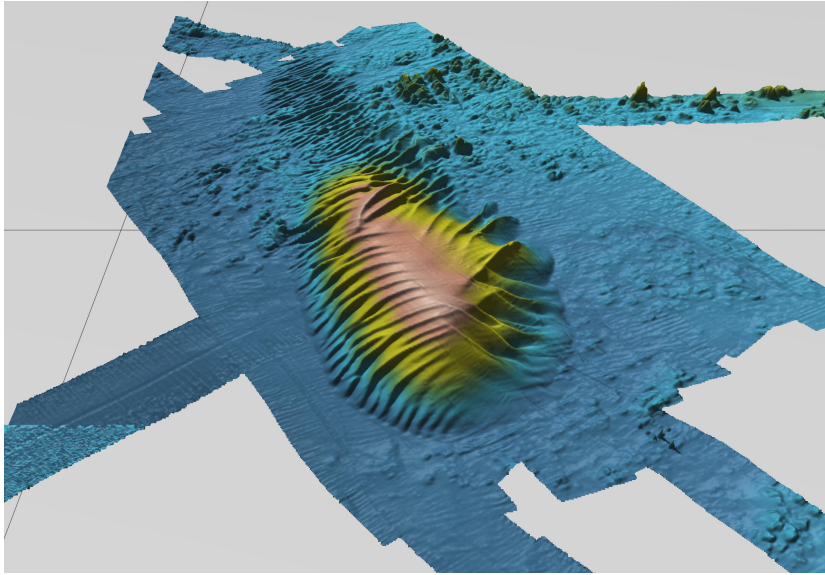


Figure 1.3: Example of an underwater Digital Elevation Model, *Dune Hydraulique*, west of Ouessant island, SHOM.

emerged recently such as the monitoring of offshore wind turbines, tidal stream generators or harbor infrastructures. Some projects of automated transport ships have also appeared in the last decade (Schiaretti, Chen, and Negenborn, 2017).

If we look closer to AUVs, they face numerous constraints and challenges because of the high required degree of autonomy. We can highlight some encountered issues:

- *Localization and communications* – One of the main problems shared by all UUVs is the localization issue. Indeed, electromagnetic waves barely propagates underwater in the ocean. Therefore, Global Navigation Satellite Systems (GNSS) cannot be used and are instead commonly replaced by acoustic based systems. They are however less accurate. For the same reasons, conventional electromagnetic wave-based communication systems do not work underwater.
- *Energy* – There is no wire that connects the AUV to a surface power source. Therefore, all the energy has to be embedded. This has a significant impact on the duration of missions, in particular when strong ocean currents have to be counteracted.
- *Depth and pressure* – AUVs are also limited in depth by mechanical constraints. High depth AUVs required thick hulls that have to resist

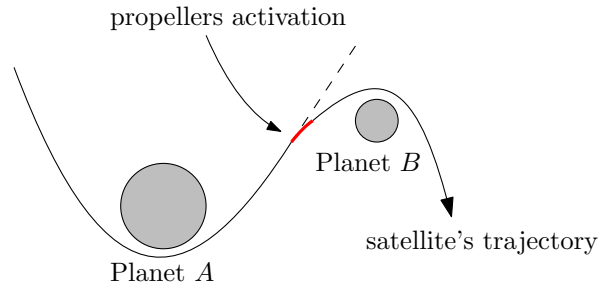


Figure 1.4: Schematic idea of the gravity assist. A short-time use of propellers allows to completely change the trajectory of the satellite.

to pressure. The size and the weight of such robots depend greatly on the maximum reachable depth, on the quantity of batteries and on the embedded sensors. In addition, the larger an AUV is, the heavier will be the launch system at surface, and therefore, the larger the size of the deployment boat will be. Optimizing the weight and size of an AUV while fulfilling the mechanical resistance constraints is then a key topic.

- *Autonomy* – An important subject of research is the autonomy of AUVs in terms of algorithms. Indeed, during a mission they mostly have to make decisions without any human intervention because of the communication issue. Moreover, the underwater environment is mostly unstructured which requires powerful algorithms to interpret sensors data, which makes it more difficult to make decisions.

### 1.1.2 The problem of drifting underwater robots

The starting point of this work was the statement that underwater robots often battling against ocean currents wasting important quantity of energy. Hence, the following question was raised: *would it be possible to use ocean currents as the main source of propulsion?* Would it also be possible, by using auxiliary propellers, to choose an opportunistic vein of current which could allow long distances to be traveled in a chosen direction?

An analogy can be drawn between underwater robots using currents and the way satellites use gravity assist maneuvers (see Figure 1.4). Indeed, to travel long distances while saving energy, space vehicles use sequentially the gravity of planets. The strategy is to use thrusters only for short and strategic moments to change the satellite's trajectory.

However, the difference between space and oceans is that currents are much less known than the position of solar system objects. Indeed, we only

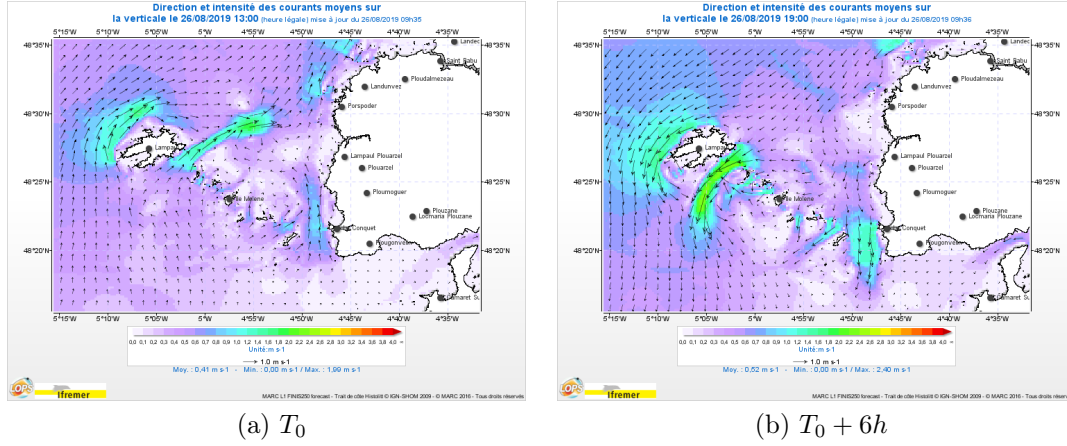


Figure 1.5: Currents in the *Fromveur passage* near Ouessant island, west from Brittany, (Ifremer MARC L1 Model – August 26, 2019, 13h00). The current reversal appears clearly between the two maps. This could enable to circumnavigate around the island.

have rough models of currents at small scales, with mesh of several hundred of meters, and only with a few days of forecast. These models (Lazure and Dumas, 2008; Pineau-Guillou, 2013) are quite good for large scales currents but become less accurate in a coastal environment with strong currents. Figure 1.5 shows for instance the result of the MARC L1 model produced by Ifremer<sup>1</sup> near Ouessant island. The mesh of the model is built upon a 250 meters wide cell. Therefore, any phenomena, such as small gyres, that are smaller than a few times the size of the cells will not appear in the results.

The current around Ouessant island are particularly interesting: indeed, in Figure 1.5, we can see that at  $T_0 + 6h$  there is a reverse of the current. It might be then possible to find trajectories such that a robot could *orbit* around Ouessant island.

**The Challenges** Assuming that we want our robot to travel from a point  $A$  to a point  $B$  following a specific path, several *Challenges* have to be solved that are non-exhaustively listed hereafter.

- *Challenge 1* – How to find a candidate path?
- *Challenge 2* – How to guarantee that the candidate path is safe and can be followed by the robot while considering uncertainties and physical limitations?

<sup>1</sup>French Research Institute for Exploitation of the Sea.

- *Challenge 3* – How to design a robot that can follow the assigned path?

In this work, we have been focusing on developing tools to answer partially the second and third items.

*Remark 1.1.* Note that the candidate *path* does not only imply a  $(x, y)$  navigation path in a  $\mathbb{R}^2$  space but can be extended to more complex spaces that could include velocity, time, energy, ... For instance, we will validate in Chapter 5 a path in a position-velocity space.

### 1.1.3 Safety of robotic systems

Guaranteeing the safety of robotic systems, i.e. that the system will never reach a dangerous state for itself or for the environment, is a key question for the use of robots. For instance, we want an autonomous car to never leave out of the road with nominal conditions. This implies to check the safety of the mechanics, the electronics, the softwares and the algorithms.

There exists numerous works that deal with safety issues with several associated softwares such as SpaceEx (Frehse et al., 2011), Acumen (Taha et al., 2016) or Astrée (Cousot et al., 2005). A significant part of the applications of these researches concerns the aviation industry where safety is critical. There has been some work to guarantee the safety of landing (Bayen et al., 2007), takeoff (Seube, Moitie, and Leitmann, 2000) or collisions avoidance (Desilles, Zidani, and Crück, 2012).

More generally, robotic systems can be represented by their state space. Guaranteeing their safety can be expressed in terms of constraints on their state space. Although the notions of paths, sets and dynamical systems will be formally introduced in Chapter 2, we can intuitively use these notions to express path constraints. For instance, we might want our system to never reach an area of the state space or at the opposite to reach at some point a specific area. Figure 1.6 illustrates a safe and an unsafe case where paths should never enter in hatched forbidden areas, should cross, at some point, a set  $\mathbb{A}$  and should finally end in a set  $\mathbb{T}$ .

If safety constraints are only expressed in the state space, the problem is time-independent and can be formalized through an Eulerian approach (Mitchell, 2007) which is often used in fluid mechanics to study stationary phenomena. Using a Lagrangian approach would have meant to study the system properties only for a finite temporal window. In this work, we have chosen to use an Eulerian approach. Indeed, we want to prove safety constraints for all time and not just for a temporal window. The reader could argue that robotic missions have a finite time and that therefore we could choose a temporal window large enough to cover the whole mission. However,

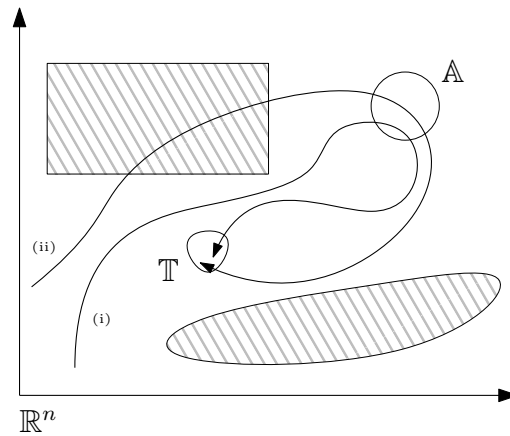


Figure 1.6: An example of paths in a  $\mathbb{R}^n$  space where hatched zones are forbidden. The paths should cross the set  $\mathbb{A}$  and end in the set  $\mathbb{T}$ . Path (i) is safe while path (ii) is not safe as it crosses a forbidden area.

for some properties such as stability or convergence, adopting an Eulerian approach is more suitable. Moreover, time can be added to the state space variables, if necessary, to still use Eulerian tools. However, this increases the dimension of the state space.

## 1.2 Contributions and Outlines

To answer the *Challenges*, we have structured this work in four main chapters. The three first chapters present theoretical tools that aim to answer *Challenge 2* while the last chapter is focusing on the design and validation of an experimental demonstrator. The guidance principle of this work is the study of paths.

In Chapter 2, we define the notions associated to dynamical systems and more particularly the notion of paths. These paths can be gathered in special sets called invariant sets. These sets will be the basic object which will be used in the next chapters. To study these sets, we will look at how they can be represented in a computer in a simpler form. Indeed, a computer cannot handle directly objects such as sets of paths. We will introduce therefore the Abstract Interpretation (AI) framework that formalizes this issue. Tools from the Interval Analysis (IA) community will be also presented to complete our toolbox. Both communities have complementary tools but few parallels have been yet drawn between them. We will try to highlight the bridges between these two areas. We will then focus on how constraints can be applied to set of paths. To this end, we will introduce the concept of Constraint



Programming. This will enable to formalize how an algorithm can solve a problem involving paths and constraints. However, we will see that existing tools do not efficiently handle paths and associated constraints. Proposing solutions to this problem is the objective of the next chapter.

In Chapter 3, we introduce the main contribution of this work which is a new domain, *i.e.* a new representation in a computer, called *Maze*, to deal with paths. This domain will be used to study the problem of bracketing largest invariant sets. We have chosen to study these specific sets as they are the cornerstone of numerous classical safety problem. We will study the properties of this new domain, and we will propose a set of tools to apply constraints to this domain. The advantages and the limitations of *mazes* will be studied in this chapter. This chapter relies on the work published in (Le Mézo, Jaulin, and Zerr, 2017a; Le Mézo, Jaulin, and Zerr, 2019).

Chapter 4 is based on results of the previous chapter. The idea is to use the bracket of invariant sets as a low level tool to solve more complex problems. We will deal with problems such as the bracket of backward and forward reachable sets, of attraction basins and of capture reach sets. These sets will be used in the next chapter to answer the *Challenges*. We will finally propose a new tool called the Eulerian state estimator that provides a framework to express and solve problems involving paths of a dynamical system. This chapter relies on the work published in (Le Mézo, Jaulin, and Zerr, 2018; Le Mézo, Jaulin, and Zerr, 2017b).

Chapter 5 addresses in a practical way the *Challenges*. We will first see how a mission involving the use of ocean currents can be formalized and how tools from previous chapters can be used for validation purposes. In a second time, we will present results around the development of a new low-cost profiling float that is specifically designed to use ocean currents. The complete robotic system will be presented with a dedicated and new depth controller. Indeed, following currents requires to precisely control the immersion depth while being energy efficient. We will use tools from the previous chapter to validate the depth controller. This chapter relies on the work published in (Le Mézo et al., 2019).

Chapter 6 concludes this work and draws up perspectives.

## 1.3 Softwares

This work has led to the development of two libraries. The first one gathers all tools that are linked to the computation of invariants and is available on *GitHub* in Python and C++ at:

<https://github.com/ThomasLeMezo/invariant-lib>.

The second one concerns the development of the embedded software around a new profiling float which has been developed for this work and is also available on *GitHub*. This software is based on the middleware ROS<sup>2</sup> at:

<https://github.com/ThomasLeMezo/seabot>.

---

<sup>2</sup>Robot Operating System (<https://www.ros.org/>)

# Chapter 2

## Tools to handle dynamical systems

### Contents

---

<b>2.1</b>	<b>Dynamical systems</b>	<b>16</b>
2.1.1	Definition	17
2.1.2	Flow map and properties of dynamical systems	18
2.1.3	Set and lattice	23
2.1.4	Invariant sets	25
2.1.5	Extension of the dynamical system model	28
2.1.6	Lyapunov theory	32
<b>2.2</b>	<b>Abstract domains</b>	<b>34</b>
2.2.1	Abstract Interpretation	36
2.2.2	Example of abstract domains	43
2.2.3	The choice of an abstract domain	56
<b>2.3</b>	<b>Constraint programming</b>	<b>57</b>
2.3.1	Constraint network and fixed points	57
2.3.2	Bracketing the solution set of a Constraint Network	62
2.3.3	Example of algorithms	68
<b>2.4</b>	<b>Conclusion</b>	<b>77</b>

---

To answer the problem of drifting underwater robots presented in the previous chapter, and in particular *Challenge 2*, we will introduce here a set of classic tools from literature. The main issue with *Challenge 2* is to determine if a path assigned to a robot is safe. As we will see, paths are complex objects to deal with. This is why, this chapter will aim to introduce conventional tools that help to handle them. These tools will also be the foundations of next chapters that will answer more specifically *Challenge 2*.

To begin, we will present the concepts of dynamical systems which will help to formalize the notion of paths and to model our problem. As we said, paths are complex objects to deal with and in particular to handle in a computer. This is why we will focus on particular set of paths: the invariant sets which are easier to deal with. Invariant sets also play a major role in the stability and safety analysis of dynamical systems which will help to prove the safety of paths. More generally, to handle sets of paths in a computer, we will introduce the framework of *Abstract Interpretation*. This framework formalizes how complex mathematical objects can be represented by a simpler object which can be handled in a computer. This will enable to make computations on simple objects while ensuring that the obtained results are still valid for the counterpart complex object. We will also need tools to translate safety constraints such as the ones described in Section 1.1.3 into constraints that can be applied to paths. We will be able to determine if a path is safe or not, *i.e.* if it fulfills all the constraints of the problem.

The chapter is divided in three parts. We will first focus on the key concepts and definitions of dynamical systems. We will then look at how these dynamical systems can be represented in a computer using the framework of *Abstract Interpretation*. We will finally deal with how dynamical system problems can be solved using *Constraint Programming*.

## 2.1 Dynamical systems

The origin of the study of dynamical systems, and more particularly their stability, is commonly recognized to be the work of Henri Poincaré at the end of the 19<sup>th</sup> century (Aubin and Dalmedico, 2002). He has laid the foundations of a crucial domain that enables to study a wide variety of systems ranging from physics and chemistry to economics.

In the case of robotics, a dynamical system formalization is a very interesting way to model robots and to study their behaviors. We will use this approach to formalize *Challenge 2* in Chapter 5.

### 2.1.1 Definition

**Definition 2.1.** A dynamical system  $\mathcal{S}$  can be defined by the following state equation (Slotine and Li, 1991):

$$\mathcal{S}: \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \quad (2.1)$$

where  $\mathbf{x}(t) \in \mathbb{R}^n$  is the state vector and  $\mathbf{f}: \mathbb{R}^n \mapsto \mathbb{R}^n$  is a non-linear Lipschitz piecewise continuous function called the evolution function of  $\mathcal{S}$ .

Using this equation, one can predict the evolution of the system starting from an initial state condition  $\mathbf{x}_0$  and an initial time  $t_0$ .

Such systems can be classified in two groups where  $\mathbf{f}$  can be explicitly time-dependent or not:

- $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t)$  is called a non-autonomous or time-variant system;
- $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$  is called an autonomous or time-invariant system.

Non-autonomous systems are generally more difficult to study than autonomous one because the initial time  $t_0$  has to be additionally taken into account. Autonomous systems are said time-invariant as the system will evolve the same way if it starts at different times but with the same initial state.

For a wide range of systems, the time dependency can be neglected and an autonomous model is sufficient.<sup>1</sup> In the case of robots, considering an autonomous model implies notably that the evolution of the environment can be neglected during the mission time.

*Remark 2.2.* In robotic problems, systems have commonly an input  $\mathbf{u}: \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^m$ . The state equation then becomes, for autonomous systems, with the evolution function  $\mathbf{f}: \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ ,

$$\mathcal{S}_{\mathbf{u}}: \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}(t), t)). \quad (2.2)$$

Note here that the input explicitly appears. If a closed-loop dynamics is considered, which means that we know the function  $\mathbf{u}$ , the system can be written as Equation (2.1). Similarly, the control can be time-dependent or only state-dependent.<sup>2</sup>

**Example 2.3.** To illustrate the previous definitions, let us consider the Van Der Pol oscillator which is a well-known and well studied dynamical system.

<sup>1</sup>In this work, unless mentioned, only autonomous systems will be considered.

<sup>2</sup>We will only consider the state-dependent case in this document where  $\mathbf{u}: \mathbb{R}^n \mapsto \mathbb{R}^m$ .

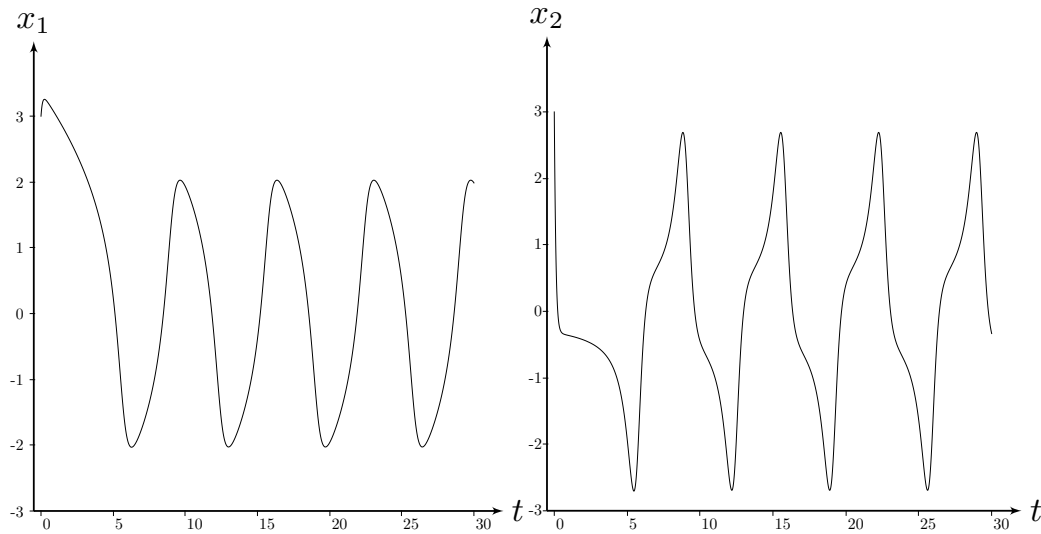


Figure 2.1: Evolution of the Van Der Pol system from an initial condition  $(\mathbf{x}_0 = (3, 3)^\top)$  in function of time.

It was discovered by Balthasar Van Der Pol in 1920. It models an electrical circuit with the following state equation:

$$\mathcal{S}_{\text{vdP}}: \begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1 \end{cases} \quad (2.3)$$

where  $\mu$  is a scalar.<sup>3</sup> This system is autonomous.

Using an initial state condition, one can simulate the evolution of the system in function of time (see Figure 2.1) or plot the evolution in a phase plane where the time does not appear directly (see Figure 2.2).

Depending on the kind of properties we want to validate, a system can be studied in the state-space domain (*i.e.* phase plane) or in the time domain.

### 2.1.2 Flow map and properties of dynamical systems

In this work, we have chosen to adopt an Eulerian point of view rather than a Lagrangian one (Mitchell, Bayen, and Tomlin, 2001). This point of view is usually adopted in fluid mechanics where fluid flow is studied instead of fluid parcel. We will then work in the state space domain rather than in the time domain. We define hereafter the different tools to handle dynamical systems with an Eulerian point of view.

<sup>3</sup>We will set  $\mu = 1$  in the rest of the document for simplicity.

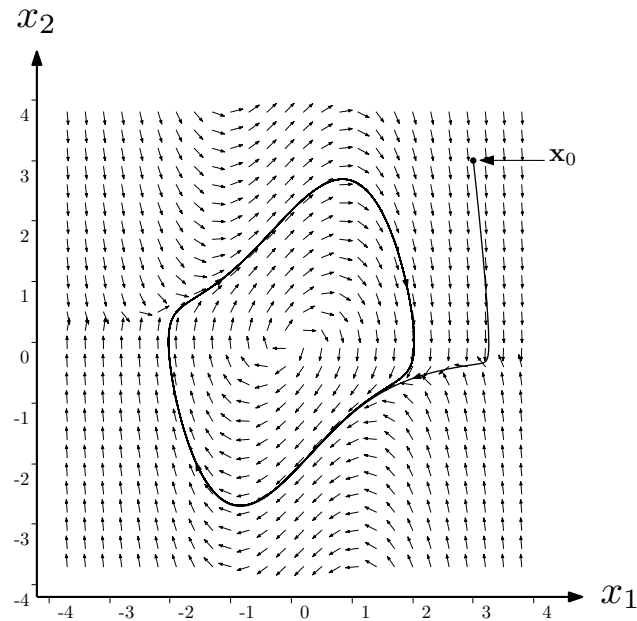


Figure 2.2: Evolution of the Van Der Pol system from an initial condition  $(\mathbf{x}_0 = (3, 3)^\top)$  in the state space.

**Definition 2.4.** The *flow map* of  $\mathcal{S}$ , denoted  $\varphi_f$  is a mapping of  $\mathbb{R} \times \mathbb{R}^n \mapsto \mathbb{R}^n$  such that for all  $\mathbf{x} \in \mathbb{R}^n$  and for all  $s, t \in \mathbb{R}$ :

$$\begin{cases} \varphi_f(0, \mathbf{x}) & = \mathbf{x} \\ \varphi_f(s, \varphi_f(t, \mathbf{x})) & = \varphi_f(s+t, \mathbf{x}) \end{cases}$$

Given an initial vector  $\mathbf{x}_0 = \mathbf{x}(0)$ , the system  $\mathcal{S}$  reaches the state  $\varphi_f(t, \mathbf{x}_0)$  at time  $t$ .

**Example 2.5.** If we take the Van Der Pol system, with the initial condition  $\mathbf{x}_0 = (3, 3)^\top$ , we obtain for instance  $\varphi_f(10.0, \mathbf{x}_0) \approx (1.932, -0.452)^\top$ . A vector field is usually used to represent the flow map of a system  $\mathcal{S}$  (see Figure 2.2).

*Remark 2.6.* In the case of a system  $\mathcal{S}_{\mathbf{u}}$  with a known input function  $\mathbf{u} \in \mathcal{U}$ , where  $\mathcal{U}$  is the set of functions from  $\mathbb{R}$  to  $\mathbb{R}^m$ , we can define similarly a flow map of  $\mathcal{S}_{\mathbf{u}}$ , denoted  $\varphi_{f, \mathbf{u}}$ .

In the case where the input function is known,  $\mathcal{S}_{\mathbf{u}}$  can be written as the previously presented dynamical system  $\mathcal{S}$ .

**Definition 2.7** (Trajectories and paths). A *trajectory* is a smooth function  $\mathbf{x}(\cdot)$  from  $\mathbb{R}$  to  $\mathbb{R}^n$ . The *path* associated with a trajectory  $\mathbf{x}(\cdot)$  is the set of all  $\mathbf{x}(t) \in \mathbb{R}^n$  where  $t \in \mathbb{R}$ , and an orientation with respect to  $t$ . We call

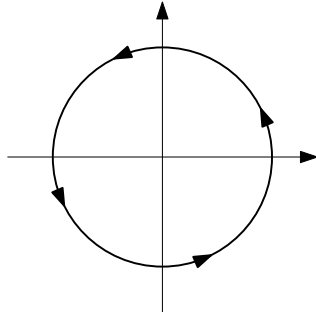


Figure 2.3: Illustration of a path corresponding to a circle with a direct trigonometric orientation.

*positive path* the path associated to all  $\mathbf{x}(t)$ ,  $t \geq 0$  and respectively *negative path* for all  $\mathbf{x}(t)$ ,  $t \leq 0$ .

**Example 2.8.** Consider the following trajectory:

$$\mathbf{x}(\cdot): \begin{cases} \mathbb{R} & \rightarrow \mathbb{R}^2 \\ t & \mapsto \begin{pmatrix} \cos(t) \\ \sin(t) \end{pmatrix}. \end{cases}$$

The trajectory is here a function of  $\mathbb{R} \mapsto \mathbb{R}^2$ . The path associated with  $\mathbf{x}(\cdot)$  is the set composed of a circle with the direct trigonometric orientation (see Figure 2.3).

*Remark 2.9* (Uniqueness of trajectories). Given an initial condition with a system  $\mathcal{S}$ , as  $\mathbf{f}$  is Lipschitz continuous, a trajectory solution  $\mathbf{x}(\cdot)$  exists and is unique (Khalil, 2002, pp. 88-90).

**Definition 2.10.** A path is *feasible* if it is associated with a trajectory  $\mathbf{x}(\cdot)$  which is solution of  $\mathcal{S}$ .

Such a path can be cyclic or can even be a single point. An important property is that a feasible path cannot make any loop, *i.e.* it cannot cross itself, or even cross another different path because of the uniqueness of the solution (see Remark 2.9). This implies that at a point  $\mathbf{x}_a$  there exists only one value for  $\dot{\mathbf{x}}$  (see Figure 2.4).

**Definition 2.11.** Let us consider a trajectory solution  $\mathbf{x}(\cdot)$  of  $\mathcal{S}$ .

- $\mathbf{x}(\cdot)$  is an *equilibrium point* if for all  $t \in \mathbb{R}$ ,  $\mathbf{f}(\mathbf{x}(t)) = \mathbf{0}$ .
- $\mathbf{x}(\cdot)$  is a *limit cycle* if there exists some  $T > 0$  such that  $\mathbf{x}(t+T) = \mathbf{x}(t)$  for all  $t \in \mathbb{R}$ . A limit cycle can be stable or unstable.



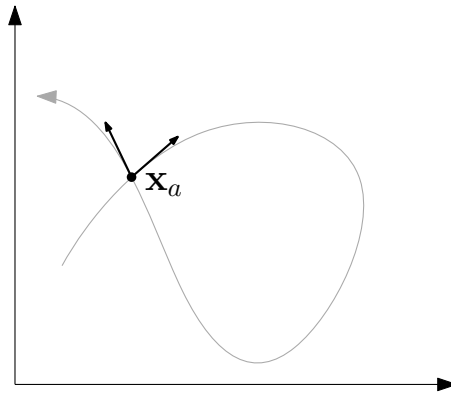


Figure 2.4: A path of  $\mathcal{S}$  cannot cross itself or another path because of the uniqueness of trajectories. The path represented here is not feasible as there exists two different values for  $\mathbf{f}(\mathbf{x}_a)$ .

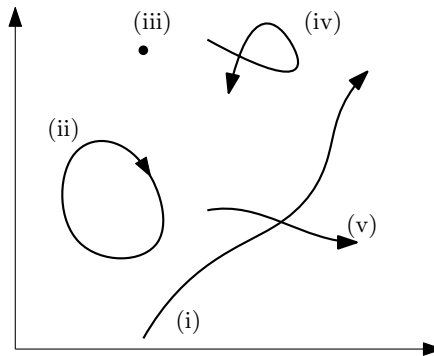


Figure 2.5: Example of feasible and unfeasible paths in the state space.

**Example 2.12.** Several paths have been represented in Figure 2.5. Let us assume that path (i) is *feasible*, then (v) cannot be *feasible* because it crosses (i). (iv) cannot be *feasible* either because it crosses itself. Paths (ii) and (iii) are special paths, respectively a cycle and a single point (or an equilibrium point).

If we consider the Van Der Pol system, it has a stable limit cycle (see Figure 2.6) whereas the reverse-timed Van Der Pol has an unstable limit cycle. There is also an equilibrium point in  $(0, 0)^\top$ .

There exists several theorems about limit cycles and equilibrium points for dynamical systems. We will only mention hereafter the *Poincaré–Bendixson* theorem<sup>4</sup> which is only true for second-order ( $\mathbb{R}^2$ ) autonomous systems.

<sup>4</sup>The wording is taken from (Slotine and Li, 1991)

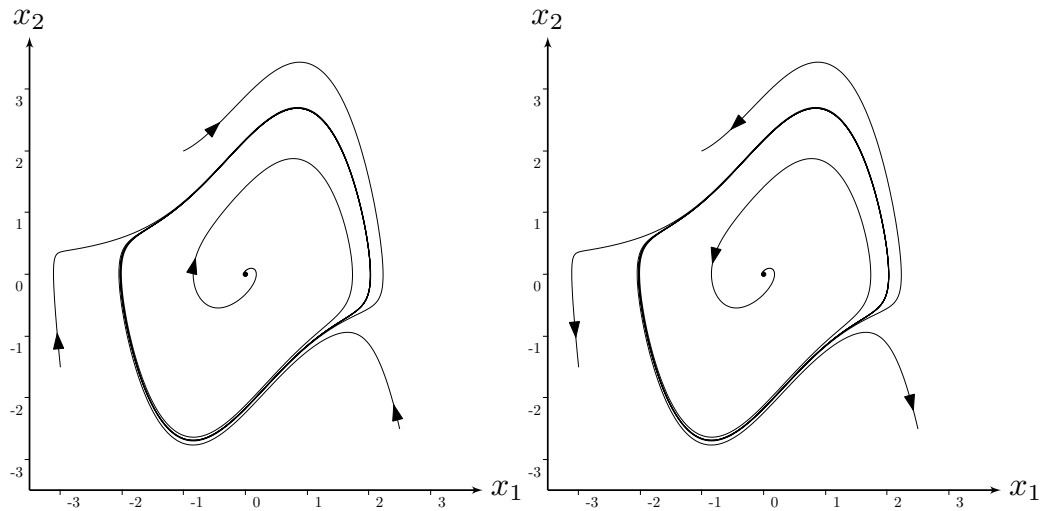


Figure 2.6: Illustration of some paths for the Van Der Pol system. The direct timed system is represented on the *left* where a stable limit cycle appears. On the contrary, on the *right*, the reverse-timed system reveals an unstable limit cycle.

**Theorem 2.13** (Poincaré–Bendixson). *If a trajectory of a second-order autonomous system remains in a finite region, the one of the following is true:*

- *the trajectory goes to an equilibrium point,*
- *the trajectory tends to an asymptotically stable limit cycle,*
- *the trajectory is itself a limit cycle.*

**Example 2.14.** For instance, if we take the trajectories that start inside the limit cycle of the Van Der Pol system, they all stay in a finite region. They tend to the limit cycle in the case of the direct-timed system and to the central equilibrium point in the case of the reverse-timed system.

*Remark 2.15.* The Lorenz system is a well known three dimensions and simple system (see Equation (2.4)) which is a counterexample of the *Poincaré–Bendixson* theorem for dimension greater than two. It was proposed in 1963 by Edward Lorenz as a simplified model of atmospheric convection. For some values of its parameters, a *chaotic* behavior appears. In that case, trajectories do not tend to a stable limit cycle but to an *attractor* (see Figure 2.7) while staying in a finite region. The trajectories of such system are particularly sensitive to initial conditions. The Lorenz system is described by the following state equations:

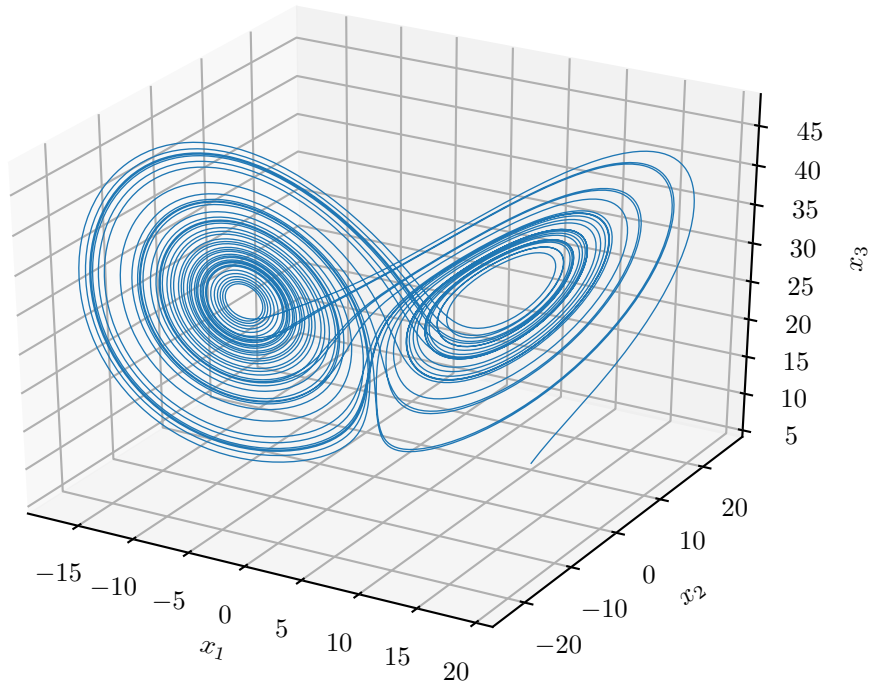


Figure 2.7: A path of the Lorenz system ( $\rho = 29$ ,  $\sigma = 9$ ,  $\beta = \frac{7}{3}$ ) with an initial condition  $\mathbf{x}_0 = (10, 10, 5)^\top$ . The trajectory tends to an attractor, the form of which recalls butterfly wings.

$$\begin{cases} \dot{x}_1 = \sigma(x_2 - x_1) \\ \dot{x}_2 = x_1(\rho - x_3) - x_2 \\ \dot{x}_3 = x_1x_2 - \beta x_3 \end{cases} \quad (2.4)$$

### 2.1.3 Set and lattice

Paths are difficult objects to individually deal with, especially with a chaotic system. One solution is to group them into sets. This will allow to study the global behavior of a dynamical system instead of a particular path with a given initial condition. The basis of the set theory has been proposed by Georg Cantor in 1874. We introduce hereafter the main concepts.

A *set* is defined as a well-defined collection of distinct objects that belong to a universe  $\Omega$ . Several operations can be applied classically to sets (see Table 2.1).

*Remark 2.16.* In addition to operations on sets, we recall the De Morgan's

Operation		
Union	$A \cup B$	$\{x \mid x \in A \text{ or } x \in B\}$
Intersection	$A \cap B$	$\{x \mid x \in A \text{ and } x \in B\}$
Difference	$A \setminus B$	$\{x \mid x \in A \text{ and } x \notin B\} = A \cap \overline{B}$
Complement	$\overline{A}$	$\{x \in \Omega \mid x \notin A\}$

(a)

Relation	
Inclusion	$A \subset B \Leftrightarrow \forall x \in A, x \in B$
Equality	$A = B \Leftrightarrow A \subset B \text{ and } B \subset A$

(b)

Table 2.1: Basic operations and relations on sets.

laws:

$$\begin{cases} \overline{A \cup B} = \overline{A} \cap \overline{B} \\ \overline{A \cap B} = \overline{A} \cup \overline{B} \end{cases} .$$

From the inclusion property, we can define a relation between sets.

**Definition 2.17** (Subset and powerset). The set  $\mathbb{X}$  is a *subset* of  $\mathbb{Y}$  if  $\mathbb{X} \subseteq \mathbb{Y}$ . We can also define the *powerset* of  $\mathbb{X}$ , noted  $\mathcal{P}(\mathbb{X})$ , as the set of all subsets of  $\mathbb{X}$ .

Functions can be extended to sets.

**Definition 2.18.** Given two sets  $\mathbb{X}$  and  $\mathbb{Y}$ , the *direct image* of a set  $\mathbb{X}_1 \subset \mathbb{X}$  under a function  $\mathbf{f}: \mathbb{X} \rightarrow \mathbb{Y}$  is:

$$\mathbf{f}(\mathbb{X}_1) = \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathbb{X}_1\}$$

The *reciprocal image* of  $\mathbb{Y}_1 \subset \mathbb{Y}$  is:

$$\mathbf{f}^{-1}(\mathbb{Y}_1) = \{\mathbf{x} \in \mathbb{X} \mid \mathbf{f}(\mathbf{x}) \in \mathbb{Y}_1\}$$

Some families of sets have specific properties on which powerful theorems can be applied. This is the case for *partially ordered sets* and *lattices* that we define hereafter. To handle sets of paths, we will try to choose families which have a lattice structure as this will enable to use useful convergence theorems.

**Definition 2.19** (Partially ordered set). A partially ordered set  $(\mathcal{A}, \leq)$  is defined by the following axioms. For all  $a, b, c \in \mathcal{A}$ :

$(\mathcal{L}, \vee, \wedge)$	
Idempotent	$a \wedge a = a$ and $a \vee a = a$
Commutative	$a \wedge b = b \wedge a$ and $a \vee b = b \vee a$
Associative	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$ and $(a \vee b) \vee c = a \vee (b \vee c)$
Absorption	$a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = a$

Table 2.2: Axioms of a *Lattice*, for all  $a, b, c \in \mathcal{L}$ .

- $a \leq a$  (reflexivity)
- if  $a \leq b$  and  $b \leq a$  then  $a = b$  (antisymmetry)
- if  $a \leq b$  and  $b \leq c$ , then  $a \leq c$  (transitivity)

**Definition 2.20** (Lattice). A *lattice*  $\langle \mathcal{L}, \leq \rangle$  is a partially ordered set, closed under least upper bound and greatest lower bound (Davey and Priestley, 2002). The least upper bound of  $x$  and  $y$  is called the *join* and is denoted by  $x \vee y$ . The greatest lower bound is called the *meet* and is written as  $x \wedge y$ .

The fundamental properties of lattices are summarized in Table 2.2.

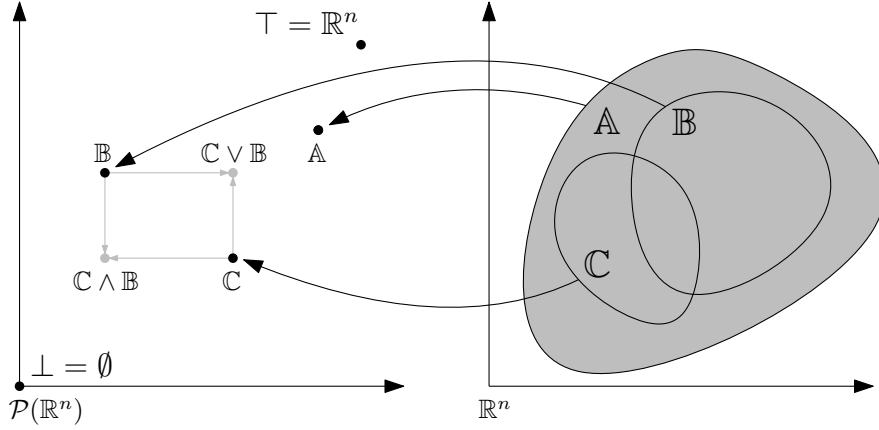
**Definition 2.21** (Complete lattice). A lattice  $\mathcal{E}$  is *complete* if for all (finite or infinite) subsets  $\mathcal{A}$  of  $\mathcal{E}$ , the least upper bound  $\bigwedge \mathcal{A}$  and the greatest lower bound  $\bigvee \mathcal{A}$  belong to  $\mathcal{E}$ . We define the top and the bottom of  $\mathcal{E}$  as  $\top = \bigwedge \mathcal{E}$  and  $\perp = \bigvee \mathcal{E}$ .

A *sublattice* of a lattice  $\langle \mathcal{L}, \leq \rangle$  is a nonempty subset of  $\mathcal{L}$  that is a lattice with the same meet and join operations  $\vee$  and  $\wedge$  as  $\mathcal{L}$ .

**Example 2.22.** The *powerset*  $\mathcal{P}(\mathbb{R}^n)$  is a complete lattice. The top element is  $\top = \mathbb{R}^n$  and the bottom element is  $\perp = \emptyset$ . In Figure 2.8, the left graph shows the powerset and the right graph shows the state space  $\mathbb{R}^n$ . We can see that we have a partial order as  $\mathbb{B} \leq \mathbb{A}$  and  $\mathbb{C} \leq \mathbb{A}$ , but  $\mathbb{B}$  and  $\mathbb{C}$  cannot be compared. The *join* and the *meet* operations are respectively the *union* and *intersection* of sets.

### 2.1.4 Invariant sets

The properties of dynamical systems can be studied through specific sets, called invariant sets, that are introduced in this subsection. They play a major role in the proof of stability of dynamical systems.

Figure 2.8: The powerset of  $\mathbb{R}^n$  is a complete lattice.

**Definition 2.23.** The set  $\mathbb{A}$  is a *positive invariant set* (or *forward invariant set*) of  $\mathcal{S}$  if for any trajectory  $\mathbf{x}(\cdot)$  solution of  $\mathcal{S}$ ,

$$\mathbf{x}(0) \in \mathbb{A}, t \geq 0 \implies \mathbf{x}(t) \in \mathbb{A}.$$

The set  $\mathbb{B}$  is a *negative invariant set* (or *backward invariant set*) of  $\mathcal{S}$  if for any trajectory  $\mathbf{x}(\cdot)$  solution of  $\mathcal{S}$ ,

$$\mathbf{x}(0) \in \mathbb{B}, t \leq 0 \implies \mathbf{x}(t) \in \mathbb{B}.$$

The set  $\mathbb{C}$  is an *invariant set* (or *forward-backward invariant set*) of  $\mathcal{S}$  if for any trajectory  $\mathbf{x}(\cdot)$  solution of  $\mathcal{S}$ ,

$$\mathbf{x}(0) \in \mathbb{C}, t \in \mathbb{R} \implies \mathbf{x}(t) \in \mathbb{C}.$$

*Remark 2.24.* Invariant sets (*positive, negative and positive-negative*) have a complete lattice structure (Kalies, Mischaikow, and Vandervorst, 2016) where the *join* operator is the union of sets  $\cup$  and the *meet* operator is the intersection of sets  $\cap$ . The top element is the whole state space  $\top = \mathbb{R}^n$  and the bottom element is the empty set  $\perp = \emptyset$ . Therefore, if  $\mathbb{A}$  and  $\mathbb{B}$  are invariant sets then  $\mathbb{A} \cup \mathbb{B}$  and  $\mathbb{A} \cap \mathbb{B}$  are also invariant sets. This allows us to define the notion of the *largest positive invariant set* enclosed in a set  $\mathbb{X}$  (see Definition 2.26 hereafter).

**Example 2.25.** In the Van Der Pol system (see Figure 2.9):  $\mathbb{C}$  the limit cycle,  $\mathbb{P}$  the equilibrium point, and  $\mathbb{I}$  the open set bounded by  $\mathbb{C}$  without  $\mathbb{P}$ , are invariant sets. We also have  $\overline{\mathbb{P}}, \overline{\mathbb{C}}, \overline{\mathbb{I}}, \{\mathbb{C} \cup \mathbb{P}\}, \{\mathbb{C} \cup \mathbb{I}\}, \{\mathbb{I} \cup \mathbb{P}\}, \{\mathbb{C} \cup \mathbb{P} \cup \mathbb{I}\}, \overline{\{\mathbb{C} \cup \mathbb{P}\}}, \overline{\{\mathbb{C} \cup \mathbb{I}\}}, \overline{\{\mathbb{I} \cup \mathbb{P}\}}, \overline{\{\mathbb{C} \cup \mathbb{P} \cup \mathbb{I}\}}, \mathbb{R}^2$  and  $\emptyset$  that are invariant sets.

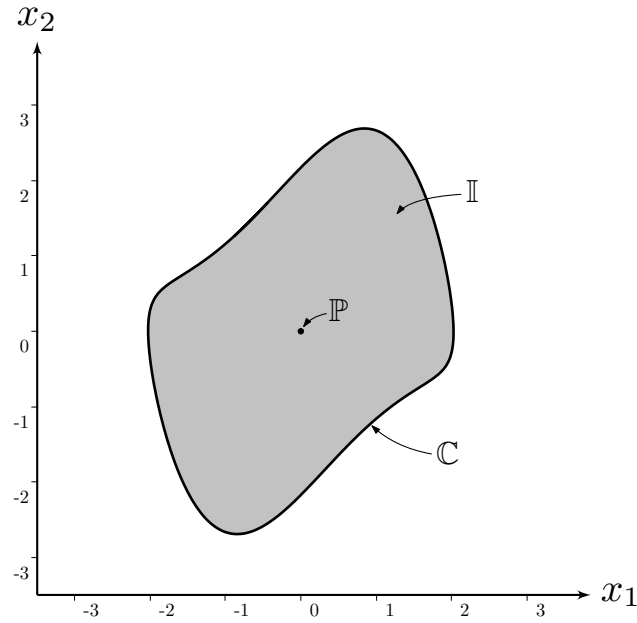


Figure 2.9: Several invariant sets can be identified in the Van Der Pol system.  $\mathbb{C}$ ,  $\mathbb{P}$ ,  $\mathbb{I}$  are invariant sets. See Figure 2.6 on page 22 for an overview of the paths.

Let us take an initial condition that belongs to one of these positive invariant sets. The associated path will stay forever inside the positive invariant set.

**Definition 2.26.** The *largest positive invariant* set of a set  $\mathbb{X}$  is:

$$\text{Inv}_f^+(\mathbb{X}) = \{\mathbf{x}_0 \mid \forall t \geq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

The *largest negative invariant* set of a set  $\mathbb{X}$  is:

$$\text{Inv}_f^-(\mathbb{X}) = \{\mathbf{x}_0 \mid \forall t \leq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

The *largest invariant* set of a set  $\mathbb{X}$  is:

$$\text{Inv}_f(\mathbb{X}) = \{\mathbf{x}_0 \mid \forall t \in \mathbb{R}, \varphi(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

*Remark 2.27.* The *largest positive or negative invariant* sets can also be defined as the union of all positive, respectively negative, invariant sets included in  $\mathbb{X}$ . Indeed, invariant sets have a lattice structure, *i.e.* there exists an upper bound. The formulas of Definition 2.26 become then a property.

*Remark 2.28.* The largest negative invariant set can be defined as the largest positive invariant set since  $\text{Inv}_f^-(\mathbb{X}) = \text{Inv}_{-f}^+(\mathbb{X})$ . We also have a relation between the largest invariant set and the intersection of the largest negative and largest positive one:

$$\begin{aligned} \text{Inv}_f(\mathbb{X}) &= \{ \mathbf{x}_0 \mid \forall t \in \mathbb{R}, \varphi(t, \mathbf{x}_0) \in \mathbb{X} \} \\ &= \{ \mathbf{x}_0 \mid \forall t \geq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X} \wedge \forall t \leq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X} \} \\ &= \{ \mathbf{x}_0 \mid \forall t \geq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X} \} \cap \{ \mathbf{x}_0 \mid \forall t \leq 0, \varphi(t, \mathbf{x}_0) \in \mathbb{X} \} \\ &= \text{Inv}_f^+(\mathbb{X}) \cap \text{Inv}_f^-(\mathbb{X}) \end{aligned}$$

Finding the largest positive invariant set can be of considerable interest in terms of safety issue. It means for instance that if the set  $\mathbb{X}$  is defined as the set where the system is safe, it is possible to find the set of all initial conditions,  $\text{Inv}_f^+(\mathbb{X})$ , into which the system can be released such that it will stay forever in  $\mathbb{X}$ . Several control problems can be rewritten as finding a largest positive invariant set.

### 2.1.5 Extension of the dynamical system model

**Differential inclusions** For some systems, the evolution function of  $\mathcal{S}$  is not known exactly but with some uncertainties. To model such systems, differential inclusions can be used. This is the case in our application where ocean currents are not known exactly: uncertainties can be modeled using sets. To this end, we can define a set-valued function that maps for instance a time-geographical position point  $(x, y, z, t)$  to a set of oceanic current values instead of single value.

**Definition 2.29.** A set-valued function  $\mathbf{F}: \mathbb{R}^n \mapsto \mathcal{P}(\mathbb{R}^m)$  maps an element of  $\mathbb{R}^n$  to a set of  $\mathbb{R}^m$ .

The definition of a dynamical system can be extended using a differential inclusion.

**Definition 2.30.** A *differential inclusion* can be written as (Blanchini and Miani, 2015):

$$\dot{\mathbf{x}}(t) \in \mathbf{F}(\mathbf{x}(t))$$

where  $\mathbf{F}: \mathbb{R}^n \mapsto \mathcal{P}(\mathbb{R}^n)$  is a set-valued function instead of a single valued function.



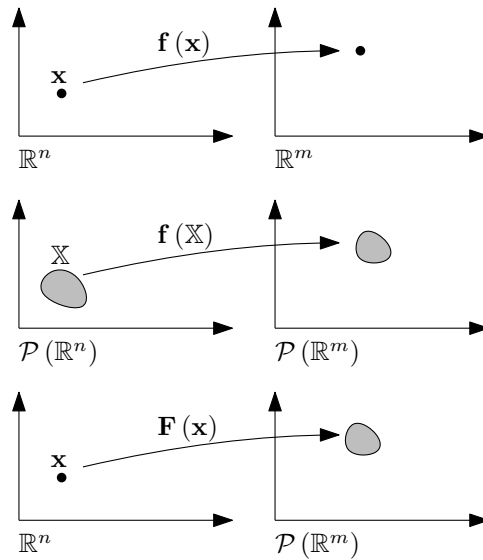


Figure 2.10: Illustration of the differences between, from top to bottom, a single value function, a set function and a set-valued function.

**Example 2.31.** Figure 2.10 illustrates the different type of functions: a single value function; a set function (Definition 2.18 on page 24) which is a simple extension of a single value function to sets; and a set-valued function (Definition 2.29 on the facing page).

**Definition 2.32.** An *uncertain dynamical system* is of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t)), \mathbf{w}(t) \in \mathcal{W}$$

In this case, the system can be modeled using a differential inclusion where  $\mathbf{F}(\mathbf{x}) = \{\mathbf{f}(\mathbf{x}, \mathbf{w}), \mathbf{w} \in \mathcal{W}\}$ .

*Remark 2.33.* The definitions of Section 2.1.4 which concern invariant sets can be extended to uncertain dynamical systems.

**Systems with input** Another case that we will consider, is systems with inputs (see Remark 2.2 on page 17). The notion of invariant sets can be in a way extended to such system by the notion of *viability*.

**Definition 2.34.** The set  $\mathbb{A}$  is a *positive viable* (or *forward viable*) set (Aubin, 2009) of  $\mathcal{S}_{\mathbf{u}}$  if

$$\forall \mathbf{x}_0 \in \mathbb{A}, \exists \mathbf{u} \in \mathcal{U}, \forall t \geq 0, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{A}.$$

The set  $\mathbb{B}$  is a *negative viable* (or *backward viable*) set of  $\mathcal{S}_{\mathbf{u}}$  if

$$\forall \mathbf{x}_0 \in \mathbb{B}, \exists \mathbf{u} \in \mathcal{U}, \forall t \leq 0, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{B}.$$

The set  $\mathbb{C}$  is a *viable* set of  $\mathcal{S}_{\mathbf{u}}$  if

$$\forall \mathbf{x}_0 \in \mathbb{C}, \exists \mathbf{u} \in \mathcal{U}, \forall t \in \mathbb{R}, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{C}.$$

We can also define the largest and the smallest viable sets.

**Definition 2.35.** The *largest positive viable* set, known as the *viability kernel* (Aubin, 2009), of a set  $\mathbb{X}$  is:

$$\text{Viab}_{\mathbf{f}}^+(\mathbb{X}, \mathcal{U}) = \{\mathbf{x}_0 \mid \exists \mathbf{u} \in \mathcal{U}, \forall t \geq 0, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

The *largest negative viable* set of a set  $\mathbb{X}$  is:

$$\text{Viab}_{\mathbf{f}}^-(\mathbb{X}, \mathcal{U}) = \{\mathbf{x}_0 \mid \exists \mathbf{u} \in \mathcal{U}, \forall t \leq 0, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

The *largest viable* set of a set  $\mathbb{X}$  is:

$$\text{Viab}_{\mathbf{f}}(\mathbb{X}, \mathcal{U}) = \{\mathbf{x}_0 \mid \exists \mathbf{u} \in \mathcal{U}, \forall t \in \mathbb{R}, \varphi_{\mathbf{f}, \mathbf{u}}(t, \mathbf{x}_0) \in \mathbb{X}\}.$$

**Example 2.36.** To illustrate these sets, let us consider the system composed of a car and an infinite straight road with a width of  $L$  (see Figure 2.11 on the facing page). The state of the car is defined in the space of its lateral position on the road ( $x$ ) and its angle with respect to the road ( $\theta$ ). We will study paths in this space. The system input is  $u \in \mathcal{U}$  the steering angle velocity of the car. We want the car to stay on the road and to move forward, which means that  $\mathbb{X} = [0, L] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ .

In Figure 2.11 on the next page we have represented with the gray hatched area what could be the complementary set of the largest positive viable set of the system. In other words, for these states, the car will leave for any input the road.

For (i), we can find for instance the input  $u = 0$  such that the car stays indefinitely on the road. For (ii), there also exists several inputs, represented in the state space by the black arrows, such that the car stays forever in the road. Indeed, the car can in a first step turns its wheels, more or less rapidly, to the right to be parallel to the road and then sets  $u = 0$ . For (iii), there exists no input such that the car does not leave the road because the angle is too important: this state is then outside  $\text{Viab}_{\mathbf{f}}^+(\mathbb{X}, \mathcal{U})$ . We can also see that if the car is parallel to the road and on the left edge, this state is inside the viable set but the driver should not turn to the left, *i.e.* we must have  $u \leq 0$ .

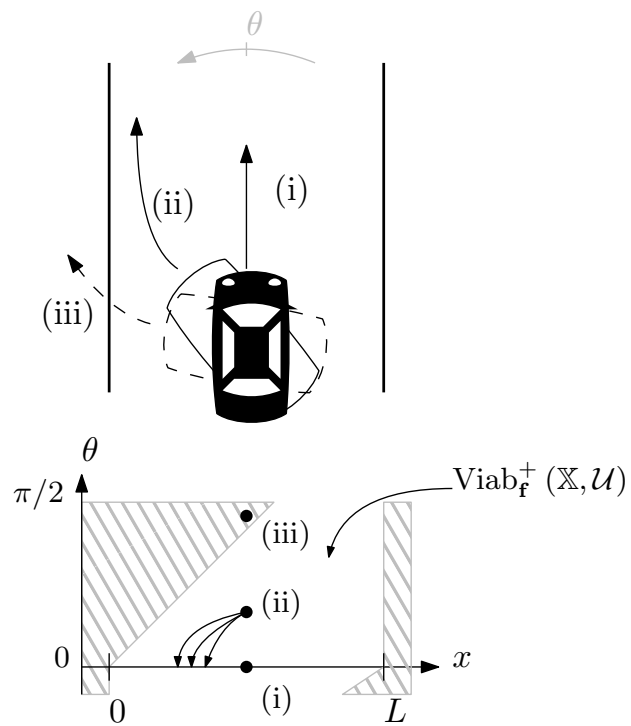


Figure 2.11: An illustrative positive viable set for the example of a car on a road.

Viable sets, as the invariant ones, play a major role in the study of dynamical systems. In Chapter 5, we will use these sets to validate the control law of our robot. We have seen in Example 2.36 on page 30 that viable sets are subsets of the state space. In the case of a car, it might be interesting to compute a viable set to detect if a distracted driver will leave the road, and consequently trigger appropriate measures.

### 2.1.6 Lyapunov theory

Traditionally, dynamical systems properties are studied through the Lyapunov theory, which was first introduced in 1892. We found then interesting to show here the advantages and drawbacks of such based methods as we will develop in the next chapters an orthogonal way to study such systems. We will introduce briefly, in this section, the main results of the theory. For more details, the reader can refer to any of the following books: (Slotine and Li, 1991; Khalil, 2002; Blanchini and Miani, 2015).

We consider here the case of an autonomous system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \quad (2.5)$$

of  $\mathbb{R}^n$  where  $\bar{\mathbf{x}} = \mathbf{0}$  is an equilibrium point which means that  $\mathbf{f}(\bar{\mathbf{x}}) = \mathbf{0}$  and  $\mathbf{f}$  is continuous. Note that by a change of variables any equilibrium point of the dynamical system can be shifted to the origin and then studied through this formalism.

**Definition 2.37.** The equilibrium point  $\bar{\mathbf{x}} = \mathbf{0}$  is *stable* if

$$\forall R > 0, \exists r > 0, \|\mathbf{x}_0\| < r \implies \forall t \geq 0, \|\varphi_{\mathbf{f}}(t, \mathbf{x}_0)\| < R.$$

It is *unstable* otherwise.

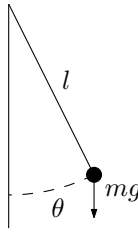
**Theorem 2.38.** Let  $\bar{\mathbf{x}} = \mathbf{0}$  be an equilibrium point for Equation (2.5) and  $D \subset \mathbb{R}^n$  be a domain containing  $\bar{\mathbf{x}} = \mathbf{0}$ . Let  $V: D \rightarrow \mathbb{R}$  be a continuously differentiable function, called a *Lyapunov function*, such that:

1.  $V(\mathbf{0}) = 0$  and  $\forall \mathbf{x} \in D \setminus \{\mathbf{0}\}, V(\mathbf{x}) > 0$ ,
2.  $\dot{V}(\mathbf{0}) = 0$  and  $\forall \mathbf{x} \in D \setminus \{\mathbf{0}\}, \dot{V}(\mathbf{x}) < 0$ ,

where  $\dot{V}$  is the derivative of  $V$  along the trajectory solution of (2.5):

$$\dot{V}(\mathbf{x}) = \left. \frac{d}{dt} V(\varphi_{\mathbf{f}}(t, \mathbf{x})) \right|_{t=0}$$

Then,  $\bar{\mathbf{x}} = \mathbf{0}$  is *stable*.

Figure 2.12: A Pendulum example with a friction coefficient  $k$ .

**Example 2.39** (Pendulum). Consider a simple pendulum (Khalil, 2002) of mass  $m$  and of length  $l$  with a friction coefficient  $k$ , we have from Newton's second law:

$$ml\ddot{\theta} = -mg \sin \theta - kl\dot{\theta}$$

By setting the state variables to  $x_1 = \theta$  and  $x_2 = \dot{\theta}$ , and the two coefficients to  $a = \frac{g}{l}$  and  $b = \frac{k}{m}$ , we obtain:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -a \sin x_1 - bx_2 \end{cases}$$

Equilibrium points are located at  $(n\pi, 0)$ ,  $n \in \mathbb{Z}$ . We try now to find a candidate Lyapunov function: let us take  $V(\mathbf{x}) = a(1 - \cos x_1) + \frac{1}{2}x_2^2$ . We have  $\dot{V}(\mathbf{x}) = a\dot{x}_1 \sin x_1 + \dot{x}_2 x_2 = ax_2 \sin x_1 + (-a \sin x_1 - bx_2)x_2 = -bx_2^2$ .

We can try to check the hypothesis of Theorem 2.38: (1) we have  $V(\mathbf{0}) = 0$  and  $V(\mathbf{x}) > 0$  for  $\mathbf{x} \in \mathbb{R}^2 \setminus \{(n\pi, 0), n \in \mathbb{Z}\}$ . So we can take for instance  $D = [-\frac{\pi}{2}, \frac{\pi}{2}] \times \mathbb{R}$ . We also have (2)  $\dot{V}(\mathbf{x}) \leq 0$  for all  $\mathbf{x} \in D$ , but this is not sufficient here to apply Theorem 2.38.

Indeed, we shall use an additional Theorem from LaSalle (LaSalle and Lefschetz, 1961) that justifies the stability of the origin. If  $\forall \mathbf{x} \in D, \dot{V}(\mathbf{x}) \leq 0$  with  $\{\mathbf{0}\} \in D$  and if no solution can stay identically in  $S = \{\mathbf{x} \in D \mid \dot{V}(\mathbf{x}) = 0\}$  except  $\mathbf{x}(\cdot) = \mathbf{0}$  then the origin is stable. With this new Theorem, we can conclude that the origin point  $(0, 0)$  is stable.

*Remark 2.40.* The asymptotic stability is also an important property to study. If the equilibrium point is asymptotically stable, we have  $\|\varphi_{\mathbf{f}}(0, \mathbf{x})\| < r \Rightarrow \lim_{t \rightarrow +\infty} (\varphi_{\mathbf{f}}(t, \mathbf{x})) = 0$  (from Definition 2.37 notations). If the second hypothesis of Theorem 2.38 is changed to  $\dot{V}(\mathbf{x}) < 0, \mathbf{x} \in D$ , then  $\bar{\mathbf{x}} = \mathbf{0}$  is asymptotically stable.

A pendulum with no friction has a stable equilibrium point which is, however, not asymptotically stable. In fact, we can find a trajectory as closed as we want from the equilibrium point that oscillate on a stable cycle.

The Lyapunov theory and positive invariant sets are closely related. *La Salle* has extended in 1961 the Lyapunov theory with *invariant sets theorems*.

**Theorem 2.41** (Local Invariant Set Theorem<sup>5</sup>). *Consider the Equation (2.5) and let  $V(\mathbf{x})$  be a scalar function with continuous first partial derivatives. Assume that:*

- for some  $l > 0$ , the region  $\Omega_l = \{\mathbf{x} \mid V(\mathbf{x}) < l\}$  is bounded,
- $\forall \mathbf{x} \in \Omega_l, \dot{V}(\mathbf{x}) \leq 0$ .

Let  $R = \{\mathbf{x} \in \Omega_l \mid \dot{V}(\mathbf{x}) = 0\}$  and  $M$  the largest positive invariant set in  $R$ . Then every solution  $\mathbf{x}(t)$  originating in  $\Omega_l$  tends to  $M$  as  $t \rightarrow \infty$ .

*Remark 2.42.* The set  $\Omega_l$  is called a level-set and is a positive invariant set.

The main drawback of Lyapunov theory is that it might be difficult to find a Lyapunov function. There is no method to find a candidate function  $V$  in a general case nor any guarantee of the existence of an analytical form of  $V$ . Moreover, the study of limit cycles is difficult with Lyapunov theory as it requires finding, most of the time, a complex function  $V$ . However, Lyapunov theory can still be used in high dimensions systems if a function  $V$  is found.

## 2.2 Abstract domains

As we have seen in the previous section, dynamical systems can be studied through specific sets such as invariant sets. In the case of ocean currents, we do not have a formal expression of the system but rather a set of numerical data. We therefore have chosen a numerical approach instead of formal computation approach to deal with invariant sets in this work.

A first step is to be able to handle in a computer these sets and perform computations. This requires to ensure that they are computer-representable. Indeed, computers do not have infinite memories and can only handle finite size objects. The problem is that most mathematical objects, we will be working with (sets of  $\mathbb{R}^n$ , paths, ...), are not representable in a computer directly. Therefore, a solution is to use instead a counterpart object which is computer-representable. The aim of this section is to study which counterpart object can be chosen and what are the links between the initial object and its counterpart.

---

<sup>5</sup>The wording is taken from (Slotine and Li, 1991)

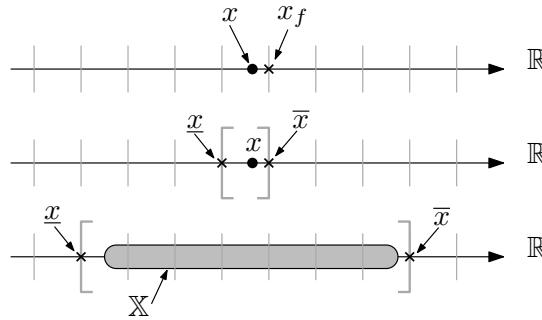


Figure 2.13: Illustration of several machine representations of a real. Gray graduations depict the reals that can be handled in a computer. For the sake of clarity, the distance between two machine's reals has been chosen constant which is not the case with the IEEE754 standard.

**Example 2.43.** To further illustrate the issue, let us consider the set of real numbers  $\mathbb{R}$ . They can be represented in a computer using the set of floating-point numbers  $\mathbb{F}$  (IEEE754 standard). In that case, only a finite number of reals can be represented as  $\mathbb{F}$  has a finite number of elements.

For a given real number  $x$ , a method is to choose the closest floating-point number  $x_f$  to represent  $x$  (see Figure 2.13). The limitation of the use of floating-point numbers is the introduction of an error between  $x_f$  and  $x$ . If computations are undertaken, the error will be propagated through the operations. This can lead to false results. For instance, let us take  $x = 0.1$ . The closest floating-point arithmetic (32-bit) to  $x$  is the number  $x_f = 0.100000001490116119384765625$ . If we now add one million time  $x_f$  to itself, we will find  $x_f + \dots + x_f = 100000.000001\dots$

In a context of safety proof of systems, if computation errors are not carefully tracked, such computations may lead to false results. One solution to guarantee the results is to bracket any real number by two bounds that are themselves representable in a computer. In other words, it means to choose a better counterpart object to represent real numbers.

**Example 2.44.** Let us consider another example with more complex data. When working with ocean currents, we would like to be able to represent the value of the currents in a certain area. Figure 2.14 is the result of an oceanic current model near the Ouessant island. The model has some uncertainties and provide data only on a discrete grid.

Let us assume that we want to handle the set of currents in the yellow box of Figure 2.14, we need to find a computer object that can represent this set. In that case, we can again use two bounds to represent the set as shown in the third graph in Figure 2.13. The set is bracketed using two representable

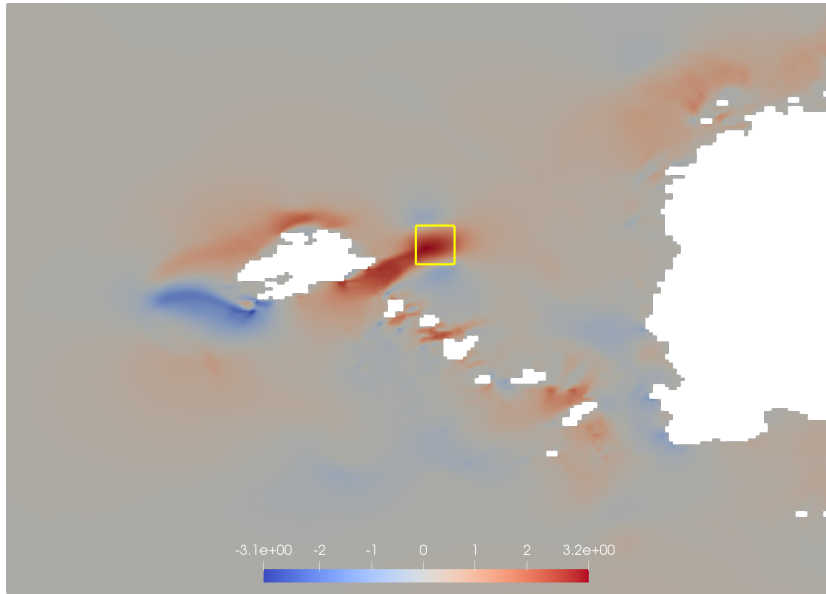


Figure 2.14: East component of the current in the *Fromveur passage* near Ouessant island, west from Brittany (24 October 2018, 00h00UTC+0, Ifremer MARC L1 Model, <http://marc.ifremer.fr>, scale in  $\text{m s}^{-1}$ ).

bounds.

To study dynamical systems, in addition to find computer representable objects, we should track computation errors carefully to guarantee the validity of results. Indeed, most dynamical systems are sensitive to initial conditions as shown in Remark 2.15 on page 22.

Two issues are then raised. How to represent a dynamical system in a computer, and how to deal with the error associated with the representation and its propagation during computation? To answer the problem, we introduce hereafter the *Abstract Interpretation* framework.

### 2.2.1 Abstract Interpretation

The link between sets and their computer representations has been studied and formalized since the late 1970s by the *Abstract Interpretation* (AI) community (Cousot and Cousot, 1977; Cousot and Cousot, 1979). The main problems addressed by the AI community were initially the static analysis, *i.e.* without execution, of computer softwares and their optimization at the stage of compilation.

In the AI community, the initial object is called the *concrete* object, and its counterpart, the *abstract* object. The *abstract* object is an *abstraction* of



the *concrete* object. Given an *abstraction*, an important issue that we must consider is that the results obtained on the *abstract* objects soundly, *i.e.* correctly, represent those which could have been obtained with the *concrete* objects. We will then be able to work only with the *abstract* objects.

**Example 2.45.** To better illustrate the following discussions, we will use in the rest of this section the example of the sets of  $\mathbb{R}^n$  as the *concrete* objects, which cannot be handled in a computer, and the boxes of  $\mathbb{R}^n$ , as the *abstract* objects.<sup>6</sup> Boxes are easy to handle in a computer and allow to perform a wide range of computations.<sup>7</sup> To use the *abstraction*, we will need to define how a box can be built from a set of  $\mathbb{R}^n$ . We will also need to check that computations on boxes soundly represent computations on sets. For instance, if we intersect two sets, does the intersection of their box abstraction soundly represent their intersection?

We first introduce the main definitions and properties of AI. We will then look at some properties that should be fulfilled by the abstractions in order to be used in a computer.

### 2.2.1.1 Definitions

**Definition 2.46.** (Cousot, 2001) Let us take  $(\mathcal{D}, \subseteq)$  and  $(\mathcal{D}^\#, \subseteq^\#)$ , be two partial ordered sets, respectively a *concrete set* and an *abstract set*.

We define two monotone functions called the *abstraction function*  $\alpha: \mathcal{D} \rightarrow \mathcal{D}^\#$  and the *concretization function*  $\gamma: \mathcal{D}^\# \rightarrow \mathcal{D}$  that map respectively all element  $x$  of  $\mathcal{D}$  to an element  $\alpha(x)$  of  $\mathcal{D}^\#$  and all element  $y$  of  $\mathcal{D}^\#$  to an element  $\gamma(y)$  of  $\mathcal{D}$ , such that:

- $\alpha$  and  $\gamma$  are increasing functions, which means that:

$$\begin{cases} \forall x, y \in \mathcal{D}, & x \subseteq y \Rightarrow \alpha(x) \subseteq^\# \alpha(y) \\ \forall x, y \in \mathcal{D}^\#, & x \subseteq^\# y \Rightarrow \gamma(x) \subseteq \gamma(y) \end{cases}.$$

- The abstract domain is an abstraction, *i.e.* an approximation, of the concrete domain:

$$\forall x \in \mathcal{D}, x \subseteq \gamma(\alpha(x)).$$

This means that  $\gamma \circ \alpha$  is extensive.

---

<sup>6</sup>Boxes of  $\mathbb{R}^n$  cannot be directly represented in a computer. However, they can be easily abstracted by boxes of  $\mathbb{F}^n$  (using floating-point numbers) that are representable (with a finite  $n$ ). We will assume that this second abstraction is implicit.

<sup>7</sup>Section 2.2.2.1 will be specifically dedicated to boxes of  $\mathbb{R}^n$ .

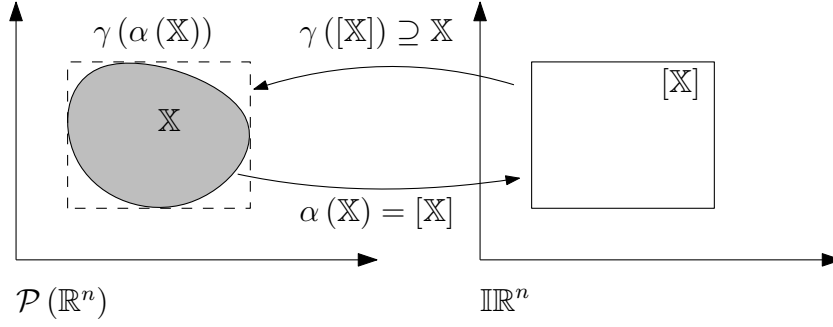


Figure 2.15: Galois connection between the powerset of  $\mathbb{R}^n$  and the powerset of  $\mathbb{IR}^n$ .

- The concrete domain does not lose any data of the abstract domain, which means that  $\forall y \in \mathcal{D}^\#, y = \alpha(\gamma(y))$  or more generally:

$$\forall y \in \mathcal{D}^\#, \alpha(\gamma(y)) \subseteq y.$$

This means that  $\alpha \circ \gamma$  is contractive.

The pair  $(\alpha, \gamma)$  defines a Galois connection denoted:  $(\mathcal{D}, \subseteq) \stackrel{\gamma}{\underset{\alpha}{\rightleftarrows}} (\mathcal{D}^\#, \subseteq^\#)$ .

**Example 2.47.** Let us take the example of sets of  $\mathbb{R}^n$  and boxes<sup>8</sup> of  $\mathbb{R}^n$ :  $(\mathcal{P}(\mathbb{R}^n), \subseteq) \stackrel{\gamma}{\underset{\alpha}{\rightleftarrows}} (\mathbb{IR}^n, \subseteq)$ . Figure 2.15 represents a set  $\mathbb{X}$  which is abstracted by a box  $[\mathbb{X}]$ . The *abstraction function*  $\alpha$  is a convex hull operator that we will define later (see Definition 2.66 on page 45). The *concretization function* is the projection of the box in  $\mathcal{P}(\mathbb{R}^n)$  as a box of  $\mathbb{R}^n$  is also a set of  $\mathbb{R}^n$ . Therefore, we can check that  $\mathbb{X} \subseteq \gamma(\alpha(\mathbb{X})) = [\mathbb{X}]$ . We also have, for any box  $[\mathbf{x}]$  of  $\mathbb{R}^n$ , that  $\alpha(\gamma([\mathbf{x}])) = [\mathbf{x}]$ . Finally,  $\alpha$  and  $\gamma$  are increasing functions.

**Example 2.48.** Let us consider a second example with two lattices (which are also partial ordered sets): the *concrete* domain  $\langle \mathcal{D}, \subseteq \rangle$  composed of six discrete states and the *abstract* domain  $\langle \mathcal{D}^\#, \subseteq^\# \rangle$  composed of only three discrete states (see Figure 2.16). We can find a pair  $(\alpha, \gamma)$  of functions that are increasing and that verify  $\forall x \in \mathcal{D}, x \subseteq \gamma(\alpha(x))$  and also, in our case, that  $\forall y \in \mathcal{D}^\#, y = \alpha(\gamma(y))$ .

We can also see the  $\alpha \circ \gamma$  is idempotent, which is a property of Galois connections.

*Remark 2.49 (Loss of information).* The abstraction generates a *loss of information* in the case where  $x \subset \gamma(\alpha(x))$ . This loss is in some way *wanted*

<sup>8</sup>As we will see in Section 2.2.2.1, the set of boxes of  $\mathbb{R}^n$  will be denoted  $\mathbb{IR}^n$ .



*Remark 2.53.* In practice, the function  $\alpha$  or  $\gamma$  might not exist for a pair of abstract and concrete domains as we will see later. In such cases we can choose to define arbitrary functions that map the abstract and concrete domains. However, we might lose some properties of the Galois connection (see (Cousot and Cousot, 1992a) for more details).

Most of the time, we only work with the abstract domain as it is the only one that can be represented in a computer. This is why we do not have to know nor to be able to compute  $\alpha$  or  $\gamma$ . However, knowing that these functions exist enables to use the properties of Galois connection and associated theorems. In particular, it allows to prove that results obtained on abstract domains are also valid for the concrete ones.

We will now consider the relation between functions in the *abstract* domain and functions in the *concrete* domain.

**Definition 2.54** (Abstract function). We consider a monotone concrete function  $f: \mathcal{D} \rightarrow \mathcal{D}$ .

- The best abstract transformer approximating  $f$  is  $f^\# = \alpha \circ f \circ \gamma$ .
- A sound abstract transformer approximating  $f$  is any operator  $f^\#: \mathcal{D}^\# \rightarrow \mathcal{D}^\#$  such that  $\alpha \circ f \circ \gamma \subseteq^\# f^\#$  (or equivalently  $f \circ \gamma \subseteq \gamma \circ f^\#$ ).

**Example 2.55.** Let us consider the example of the best abstract transformer  $f^\#$  of a function  $f$ , with the Galois connection  $(\mathcal{P}(\mathbb{R}^n), \subseteq) \xrightleftharpoons[\alpha]{\gamma} (\mathbb{I}\mathbb{R}^n, \subseteq)$ , that contracts<sup>9</sup> sets of  $\mathbb{R}^n$ . Figure 2.17 shows, on the top left, a set  $\mathbb{X}$  which is contracted into a set  $\mathbb{Y}$ . On the top right, we have  $[\mathbb{X}]$  the box abstraction of  $\mathbb{X}$ . The best abstract transformer  $f^\#$  is used to obtain  $[\mathbb{Y}] = f^\# \circ \alpha(\mathbb{X})$ . We can finally obtain  $\mathbb{Z} = \gamma \circ f^\# \circ \alpha(\mathbb{X})$ . We note that  $\mathbb{Y} = f(\mathbb{X}) \subset \mathbb{Z}$  which is only due to the *loss of information* of the abstraction.

In the case where we only have a sound abstract transformer approximating  $f$ , we will have obtained an over approximation of the set  $\mathbb{Z}$ . In this case the *loss of information* is due to the abstraction and to the abstract transformer.

*Remark 2.56.* From Example 2.55, we highlight that in the case where we only have a sound abstract transformer approximating  $f$ , a *loss of information* appears. This phenomenon is not due to the abstraction but to the approximation of the function  $f$  in the abstract domain. Therefore, we have two possible sources of *loss of information*: from the abstract transformer and from the abstract function (see Remark 2.49).

<sup>9</sup>The notion of contractor is defined later in this chapter (see Definition 2.105 on page 62).

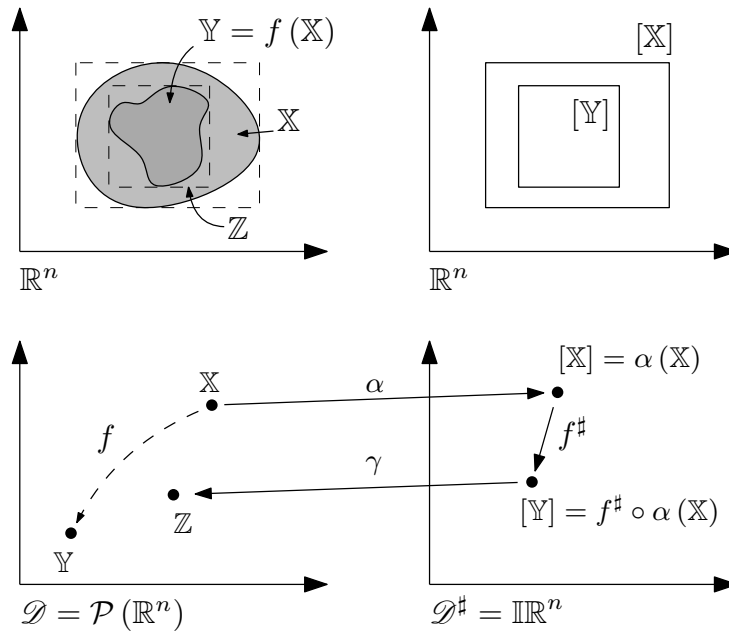


Figure 2.17: Illustration of the use of the best abstract transformer  $f^\#$  of a function  $f$  that contracts sets of  $\mathbb{R}^n$ . The abstract domain is the set of boxes of  $\mathbb{R}^n$ . There is a *loss of information* which is only due to the abstraction.

In practice, computing the best abstract transformer can be complex and time-consuming, this is why the use of a sound abstract transformer could be sufficient.

### 2.2.1.2 Properties of abstract domains

As said previously, an abstract domain should be representable in a computer, but this is not sufficient. Indeed, we will look here at some additional properties that an abstract domain should fulfill to avoid convergence issues of algorithms. We will deal with *closure operators*, *Ascending Chain Condition* and *widening operators*.

**Definition 2.57.** A *closure operator* on a set  $\mathbb{S}$  is a function  $\rho: \mathcal{P}(\mathbb{S}) \rightarrow \mathcal{P}(\mathbb{S})$  such that for  $X$  and  $Y$  in  $\mathcal{P}(\mathbb{S})$ :

- $X \subseteq Y \implies \rho(X) \subseteq \rho(Y)$  (monotonic)
- $X \subseteq \rho(X)$  (extensive)
- $\rho(\rho(X)) = \rho(X)$  (idempotent)

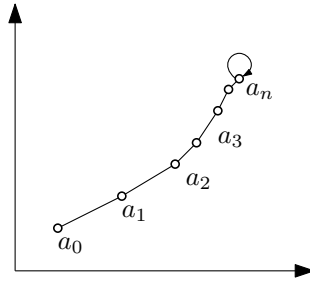


Figure 2.18: Ascending chain condition in a complete lattice. There exists an  $n$  such that the sequence is constant.

The image of a closure operator is closed by (infinite) intersection. This operator is also called a *hull operator* and the family of sets that have a closure operator is called a *Moore family*.

Another important property that should be looked at is the *Ascending Chain Condition* (ACC) that we introduce below. In particular, verifying this property is a key point to validate the convergence of algorithms in a finite number of steps. This is mandatory when using computers. This property is stronger than the existence of a closure operator. Sets that do not have a closure operator will not verify the ACC.

**Definition 2.58** (Ascending Chain Condition). *A partially ordered set  $(\mathcal{A}, \leq)$  satisfies the ascending chain condition (ACC) if for any weakly ascending sequence  $(a_i)_{i \in \mathbb{N}}$ , i.e.  $a_0 \leq a_1 \leq a_2 \leq \dots$ , there exists a positive integer  $n$  such that  $a_n = a_{n+1} = \dots$  (see Figure 2.18). Equivalently, this means that for any non-empty subset  $\mathcal{B} \subseteq \mathcal{A}$ ,  $\mathcal{B}$  has at least one maximal element.*

A descending chain condition can be similarly defined with a weakly descending sequence.

**Example 2.59.** The set of float numbers  $\mathbb{F}$  used in computers equipped with the usual order relation verifies the ascending chain condition. Indeed,  $\mathbb{F}$  has a finite number of elements, so in the worst case one of the two bounds is reached in a finite number of steps. This is also the case for boxes of  $\mathbb{F}^n$ . The set of natural numbers  $\mathbb{N}$  verifies the descending chain condition but does not verify the ascending one as it has a lower bound but no upper bound.

In particular, it means that if we apply several times a *contractor operator*<sup>10</sup>  $\mathcal{C}$  on a box of  $\mathbb{F}^n$ , there exists a number of iterations  $n$  such that the box is no more contracted on additional iterations.

<sup>10</sup>see Footnote 9 and Section 2.3 on convergence proof.

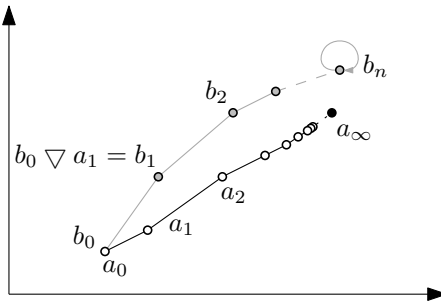


Figure 2.19: Illustration of the widening operator.

In some cases, partially ordered sets  $(\mathcal{A}, \leq)$  do not verify the ascending chain condition. Indeed, it means that some sequences of calculations are not stationary. This is a major issue when a fixed point has to be reached with a computer (see later the Section 2.3). This is why a *widening operator* is introduced to counteract the issue.

**Definition 2.60** (Widening operator (Cousot and Cousot, 1992b)). A *widening operator*  $\nabla: \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$  is defined in order to verify the following conditions:

- $\forall x, y \in \mathcal{D}, x \subseteq x \nabla y$  and  $y \subseteq x \nabla y$
- For all increasing chain  $(a_i)_{i \in \mathbb{N}}$  of elements of  $\mathcal{D}$ , i.e.  $a_0 \subseteq a_1 \subseteq \dots$ , the increasing chain  $(b_i)_{i \in \mathbb{N}}$  defined by 
$$\begin{cases} b_0 & = a_0 \\ b_{i+1} & = b_i \nabla a_{i+1} \end{cases},$$
 is not strictly increasing.

*Remark 2.61.* The widening operator can be seen as an over approximation of the union. This allows to obtain an over approximation of the limit of an increasing chain (see Figure 2.19). An increasing chain  $(a_i)$  is over-approximated by an increasing chain  $(b_i)$  that is stationary after a number of iterations. Partially ordered sets that do not verify the ACC can be still used inside algorithms, while keeping the property of convergence in a finite number of steps. This operator will be particularly useful when using convex polytopes.

Note that a *narrowing operator* that produces an over-approximation of the intersection can also be defined.

### 2.2.2 Example of abstract domains

To validate the behavior of a robot drifting in the currents, we have to study the paths of a dynamical system. A first level of abstraction was to use

invariant sets that are sets of  $\mathbb{R}^n$  instead of paths (see Definition 2.26). However, working with sets is still difficult as they cannot be represented in a computer. We will present hereafter well-known domains that can abstract sets of  $\mathbb{R}^n$ .

Several abstract domains have been formalized in the AI community such as intervals (Cousot and Cousot, 1976), simple congruences, linear equalities, linear congruences, polyhedra (Cousot and Halbwachs, 1978), octagons (Miné, 2006), ellipsoids, varieties, etc. In the following, we will focus on two of them: intervals and convex polytopes. We will also introduce subpavings that are specific sets of intervals.

### 2.2.2.1 Intervals

The formalization of intervals with the associated interval arithmetic has started in the mid 1960s slightly before and in parallel with its development in the AI community. Ramon E. Moore published in 1966 a reference book entitled *Interval Analysis* (IA) which is considered as one of the reference in the field (Moore, 1966). This work was driven by the need to handle numerical computer errors (see Example 2.43). Nowadays, interval arithmetic is a useful tool for the validation of safety constraints (Gouttefarde, Daney, and Merlet, 2011; Daney, Papegay, and Neumaier, 2004).

We will attempt in this section to adopt a view that bring together the tools from the IA and AI communities. Indeed, it seems to exist very few links between them although their tools are quite similar. Most of the definitions and properties of intervals will be given from the IA point of view while highlighting the links to AI.

#### Definitions and arithmetic

**Definition 2.62.** We define an interval  $[x]$  as a closed and connected subset of  $\mathbb{R}$ . The set of all intervals is denoted by  $\mathbb{IR}$ . We have:

$$[x] = [\underline{x}, \bar{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \bar{x}\}$$

where  $\underline{x}$  is the lower bound and  $\bar{x}$  is the upper bound.

*Remark 2.63.* The empty set  $\emptyset$  is considered as an interval which represents the absence of solution. This allows the set of intervals  $\mathbb{IR}$ , to be closed with respect to the intersection. When an interval contains only a single element, it is called a *singleton* or a *degenerated interval*.

An important point is that if the bounds of intervals are subsets of  $\mathbb{F}$  (set of floating-point numbers), then the domain verifies the ACC (see Definition 2.58). This implies that monotonic operators working with intervals



will converge toward a limit in a finite number of steps without the need of a widening operator (see also Example 2.59).

**Example 2.64.**  $[-4, 2], [-\infty, \infty], [-\infty, 4], \{2\}$  are all intervals of  $\mathbb{IR}$ .

The East component of the currents in the yellow box in Figure 2.14 on page 36, can be for instance enclosed in the interval  $[-0.244, 3.186] \text{ m s}^{-1}$ . We already mentioned that the forecast of ocean currents is subject to important uncertainties. Intervals are here an interesting way to enclose uncertainties of data while maintaining the guarantee of computations. However, this requires to have known bounded uncertainties. For instance, if the ocean current model has a known bounded velocity error of 10%, the currents in the yellow box can be enclosed within the interval  $[-0.248, 3.505] \text{ m s}^{-1}$ .

**Definition 2.65.** An *interval vector*, or a box,  $[\mathbf{x}]$  of  $\mathbb{R}^n$  is defined as the Cartesian product of  $n$  intervals. The set of all interval vectors of  $\mathbb{R}^n$  is denoted  $\mathbb{IR}^n$ . We have:

$$[\mathbf{x}] = [x_1] \times [x_2] \times \cdots \times [x_n].$$

The notion of *singleton* or *degenerated interval* can be extended to interval vectors. They are called *degenerated* if one dimension is at least a *degenerated interval*, and *singleton* when all dimensions are degenerated.

**Definition 2.66.** The *interval hull*, or *box hull*, of a subset  $\mathbb{A} \subset \mathbb{R}^n$ , denoted  $[\mathbb{A}]$ , is the smallest box of  $\mathbb{IR}^n$  that contains  $\mathbb{A}$ .

*Remark 2.67.* The interval hull operator that associates the set  $\mathbb{A} \subset \mathbb{R}^n$  to  $[\mathbb{A}]$  is called a *wrapper* in the IA community. This operator is the *abstraction function*  $\alpha$ , of the AI community. The sets of  $\mathbb{R}^n$  and the boxes of  $\mathbb{R}^n$  form a Galois connection  $(\mathcal{P}(\mathbb{R}^n), \subseteq) \stackrel{\gamma}{\dashv} \underset{\alpha}{(\mathbb{IR}^n, \subseteq)}$ . This operator is then the best abstraction, *i.e.* it optimally abstracts sets of  $\mathbb{R}^n$  (see Theorem 2.51 on page 39).

The interval hull operator is also a closure operator (see Definition 2.57 on page 41). Indeed, for two sets  $\mathbb{X}, \mathbb{Y} \in \mathcal{P}(\mathbb{R}^n)$ , we can trivially verify that: if  $\mathbb{X} \subseteq \mathbb{Y}$  then  $[\mathbb{X}] \subseteq [\mathbb{Y}]$ ,  $\mathbb{X} \subseteq [\mathbb{X}]$  and  $[[\mathbb{X}]] = [\mathbb{X}]$ . In particular, it means that an infinite intersection or union of boxes is still a box.

**Definition 2.68.** The *width* of an interval vector  $[\mathbf{x}]$  is defined as

$$w([\mathbf{x}]) = \max_{1 \leq i \leq n} w([x_i])$$

and  $w_i([\mathbf{x}]) = w([x_i])$ .

Operation		
Intersection	$[x] \cap [y]$	$\{z \in \mathbb{R} \mid z \in [x] \wedge z \in [y]\}$
Union	$[x] \cup [y]$	$\{z \in \mathbb{R} \mid z \in [x] \vee z \in [y]\}$
Interval Union (interval hull)	$[x] \sqcup [y]$	$[[x] \cup [y]]$
Interval Difference	$[x] \setminus [y]$	$\{x \in [x] \wedge x \notin [y]\}$

Table 2.3: Set operations on Intervals.

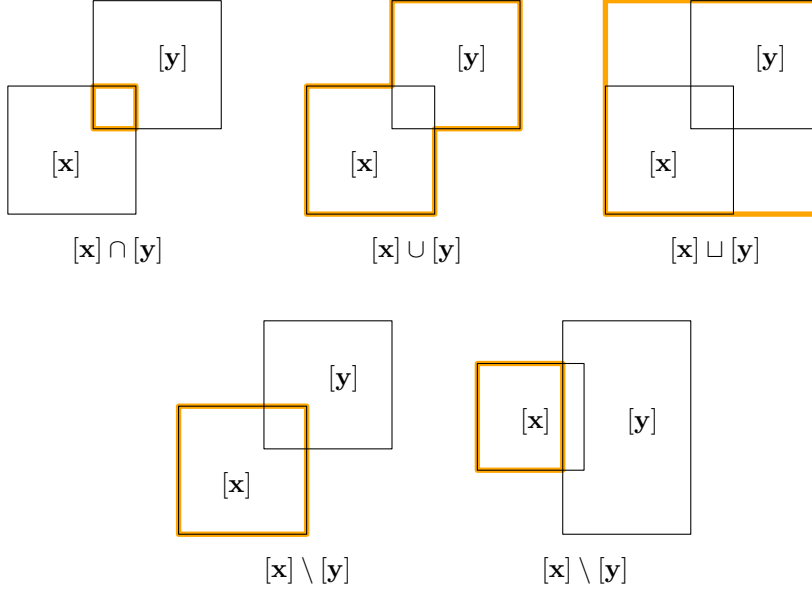


Figure 2.20: Examples of set operations on Intervals.

The *lower bound*  $\underline{x}$  and the *upper bound*  $\bar{x}$  of an interval vector  $[x]$  are defined as the following punctual vectors:

$$\begin{aligned}\bar{x} &= (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)^\top, \\ \underline{x} &= (\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n)^\top.\end{aligned}$$

**Definition 2.69.** Interval counterpart of classic set operations are defined in Table 2.3. They are the best abstract transformers of the set operators in the interval domain.

*Remark 2.70.* We can observe that the intersection of two intervals is still an interval ( $\mathbb{IR}$  is closed by intersection). It is not always the case for the classic union  $\cup$ . This is why the *interval union*  $\sqcup$  is defined as the interval hull of the union of two intervals (see examples on Figure 2.20). The *loss of information* is due to the abstraction.

If we associate to the set  $\mathbb{IR}^n$ , the intersection  $\cap$  as the *meet* operation, and the interval union  $\sqcup$  as the *join* operation, we obtain a complete lattice structure. The top element is  $\top = \mathbb{R}^n$  and the bottom element is  $\perp = \emptyset$ . This lattice structure property will be essential in Section 2.3 to prove the convergence of algorithms.

**Definition 2.71.** An arithmetic of intervals can be defined:

$$[x] \diamond [y] = [\{x \diamond y \in \mathbb{R} \mid x \in [x], y \in [y]\}]$$

where  $\diamond \in \{+, -, \cdot, /\}$ .

**Example 2.72.** Let us consider several operations:

- $[1, 4] + [-4, 2] = [-3, 6]$ ,
- $[1, 4] - [-4, 2] = [-1, 8]$ ,
- $[1, 4] \times [-4, 2] = [-16, 8]$ ,
- $[1, 4] / [-4, 2] = [-\infty, +\infty]$ ,
- $[1, 4] / [2, 4] = [0.25, 2]$ ,
- $[1, 4] / [0] = \emptyset$ .

We can note that the division by an interval which contains zero is allowed as the bounds can be set to  $\pm\infty$ .

**Interval functions and inclusion functions** We will show now, how functions of  $\mathbb{R}^n$  can be abstracted in the interval domain. In the IA community, the abstracted function  $f^\sharp$  of a function  $f$  is denoted  $[f]$ .

**Definition 2.73.** Classic functions can be abstracted by *interval functions*. Let  $f$  be a function from  $\mathbb{R}$  to  $\mathbb{R}$ , the associated interval function  $[f]$  is defined as:

$$[f]([x]) = [\{f(x) \mid x \in [x]\}].$$

**Example 2.74.** Let consider several classic functions:

- $[\text{exp}]([1, 2]) = [2.718, 7.389]$ ,
- $[\text{sqr}]([2, 4]) = [4, 16]$ ,
- $[\text{sqr}]([-4, 2]) = [0, 16]$ ,

- $[\arctan]([-2, 4]) = [-1.107, 1.326]$ ,
- $[\sin]([\frac{3\pi}{4}, \frac{7\pi}{3}]) = [-1, \frac{\sqrt{3}}{2}]$ .

*Remark 2.75.* The build of classic functions is straightforward for monotonic functions as they can be expressed in terms of bounds. For instance, the arctangent function is monotonic positive so:  $[\arctan]([x]) = [\arctan(\underline{x}), \arctan(\overline{x})]$ .

This is more difficult for non-monotonic functions such as sine or cosine where modulo operations have to be performed. Nevertheless, all classic functions have nowadays efficient algorithms that evaluate their associated interval functions. This has been the subject of important research efforts (“IEEE Standard for Interval Arithmetic” 2015; Goualard, 2015).

**Definition 2.76.** More generally, an interval function  $[\mathbf{f}]$  from  $\mathbb{IR}^n$  to  $\mathbb{IR}^m$  is an *inclusion function* of the function  $\mathbf{f}$  from  $\mathbb{R}^n$  to  $\mathbb{R}^m$  (see Definition 2.18) if:

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) \subseteq [\mathbf{f}]([\mathbf{x}]).$$

An inclusion function  $[\mathbf{f}]$  is said to be *minimal*, and denoted  $[\mathbf{f}]^*$ , if for any  $[\mathbf{x}]$ ,  $[\mathbf{f}]([\mathbf{x}])$  is the smallest box that contains  $\mathbf{f}([\mathbf{x}])$ .

It is *convergent* if for any sequence of boxes  $[\mathbf{x}](k)$ ,

$$\lim_{k \rightarrow \infty} w([\mathbf{x}](k)) = 0 \Rightarrow \lim_{k \rightarrow \infty} w([\mathbf{f}]([\mathbf{x}](k))) = 0.$$

*Remark 2.77.* If  $\mathbf{f}$  is monotone, then  $[\mathbf{f}]$  is a sound abstract transformer approximating  $\mathbf{f}$ , and  $[\mathbf{f}]^*$  is the best abstract transformer approximating  $\mathbf{f}$  (see Definition 2.54 on page 40). The use of a minimal inclusion function will therefore reduce the *loss of information*.

*Remark 2.78* (Natural inclusion functions). A simple way to build an *inclusion function* is to replace each classic functions, variables and operators by their interval counterparts as defined previously. The obtained formal expression is called the *natural inclusion function*. In most cases, this function is not minimal except if all operators and elementary functions are continuous and each variable appears only once in the formal expression. A rewriting of the formal expression to satisfy this condition is one of the techniques to obtain a *minimal inclusion function* (Jaulin et al., 2001).

Limiting the pessimism of inclusion functions has been the subject of numerous works and implementations based on centered form, Taylor extensions, ... In this work, we have been using the *IBEX library*<sup>11</sup> to deal with

<sup>11</sup><http://www.ibex-lib.org/>

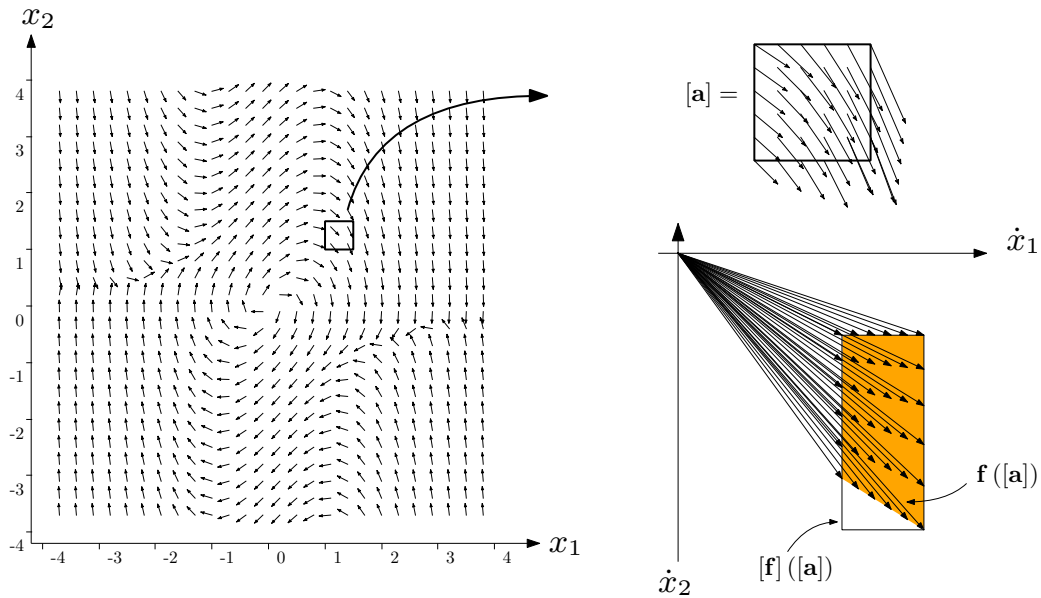


Figure 2.21: Evaluation of an inclusion function build from the Van Der Pol system. On the left: the normalized vector field. On top right: a zoom on  $[\mathbf{a}]$ , and on the bottom right, the sets  $\mathbf{f}([\mathbf{a}])$  and  $[\mathbf{f}]([\mathbf{a}])$  in the space  $(\dot{x}_1, \dot{x}_2)$ .

intervals and inclusion functions. We assume that the issue of function evaluations is solved by the library and that it gives guaranteed and sufficient accurate results.

Moreover, we will assume, in this work, that all inclusion functions are convergent as this property will be required to prove the convergence of algorithms.

**Example 2.79.** We can build an inclusion function from the Van Der Pol system (presented on page 17). If we evaluate the function (see Figure 2.21) on the interval vector  $[\mathbf{a}] = [1, 1.5] \times [1, 1.5]$ , we get the black hull box  $[\mathbf{f}]([\mathbf{a}])$  equal to  $[1, 1.5] \times [-3.375, -1]$ . We can see here that we verify  $\mathbf{f}([\mathbf{a}]) \subset [\mathbf{f}]([\mathbf{a}])$  where  $\mathbf{f}([\mathbf{a}])$  is the orange-colored set. The inclusion function is here convergent. We have also a monotonic function: for instance, if we take  $[\mathbf{b}] = [1.1, 1.2] \times [1.1, 1.2]$ , we have  $[\mathbf{f}]([\mathbf{b}]) = [-1.728, -1.331] \subset [\mathbf{f}]([\mathbf{a}])$ .

*Remark 2.80 (Wrapping effect).* As seen on Example 2.79, the use of boxes produces an over-approximation of  $\mathbf{f}([\mathbf{x}])$ . This phenomenon is called the *wrapping effect* in the IA community and is equivalent to the *loss of information* due to the abstract transformer described in Remark 2.49. When functions are composed, the over-approximation is propagated and increases along each evaluation which can lead to accuracy issues.

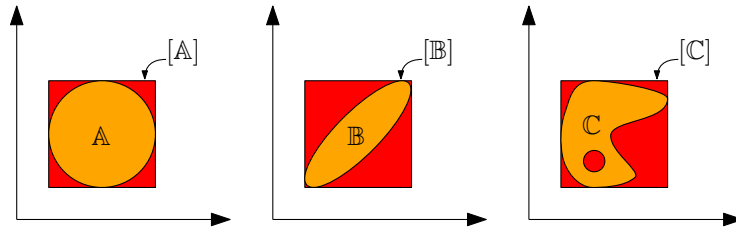


Figure 2.22: Illustration of the *wrapping effect* resulting from the use of the convex hull operator. This effect is called the *loss of information* in the Abstract Interpretation community.

To summarize, there are two kinds of *wrapping effect*: the one described above which is due to the use of an inclusion function (Remark 2.56), and the one which is due to the abstraction (Remark 2.49). Figure 2.22 recalls how a *loss of information* or a *wrapping effect* may appear when sets of  $\mathbb{R}^n$  are enclosed by boxes.

Several techniques can be used to reduce the *wrapping effect* such as bisecting boxes, but the pessimism remains inherent to the interval representation.

### 2.2.2.2 Convex Polytopes

To limit the interval wrapping effect, convex polytopes can be used. They indeed better approximate sets of  $\mathbb{R}^n$ . However, there are some drawbacks that we will highlight hereafter.

**Definition 2.81.** A convex polytope  $\mathbf{P}$  can be defined as the intersection of a set of half-spaces or the combination of linear inequality constraints. More formally, in an euclidean space  $\mathbb{R}^n$ , let us consider  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$ :

$$\mathbf{P} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{Ax} \leq \mathbf{b}\}.$$

We denote  $\mathbb{PR}$  the set of all convex polytopes.

**Example 2.82.** Let us take  $\mathbf{P} \in \mathbb{PR}^2$ ,  $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 2 \\ -1 & -1 \\ 1 & -1 \\ -10 & -1 \end{pmatrix}$  and  $\mathbf{b} =$

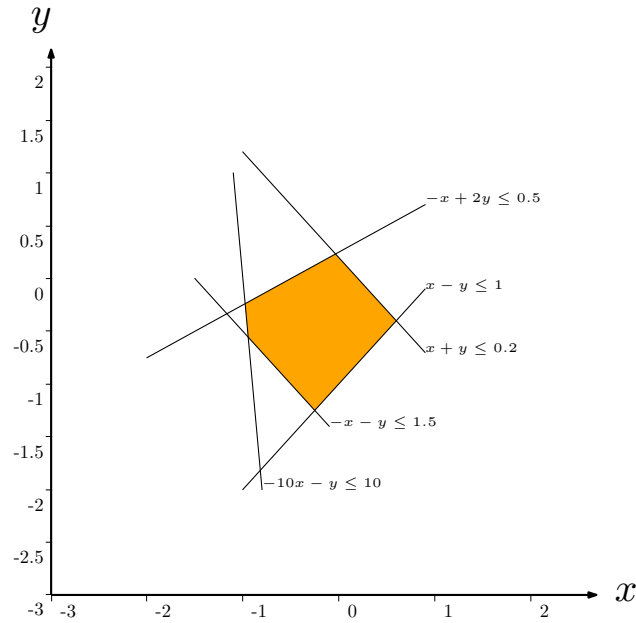


Figure 2.23: Example of a two-dimensional polytope.

$$\begin{pmatrix} 0.2 \\ 0.5 \\ 1.5 \\ 1 \\ 10 \end{pmatrix}$$
. The corresponding polytope  $\mathbf{P}$  is represented in Figure 2.23.

*Remark 2.83.* In the general case, using polytopes will limit the *loss of information*. They allow to obtain a better over-approximation of a set compared to a box which is indeed a specific case of polytope. Nevertheless, this is achieved at the cost of an increase on computer's memory space as the number of constraints ( $m$ ) is greater than with boxes.

**Definition 2.84.** (Halbwachs, Proy, and Roumanoff, 1997) The intersection  $\cap$  of two convex polytopes  $\mathbf{P}, \mathbf{Q} \in \mathbb{P}\mathbb{R}^n$  is defined as the polytope whose linear inequality constraint system is the conjunction of those of  $\mathbf{P}$  and  $\mathbf{Q}$ .

The convex hull  $\sqcup$  is defined as the least convex polytope containing both  $\mathbf{P}$  and  $\mathbf{Q}$ . Note that, as for intervals, the union  $\cup$  of two convex polytopes is not necessary a convex polytope (see Figure 2.24).

*Remark 2.85.* The number of linear constraints of the resulting polytope after applying intersections or after applying a convex hull operator will increase in most of the cases. Constraints can be sometimes removed with intersections as they are redundant but not in the general case. Limiting this phenomenon is a key factor to allow any algorithm to be implemented in a computer.

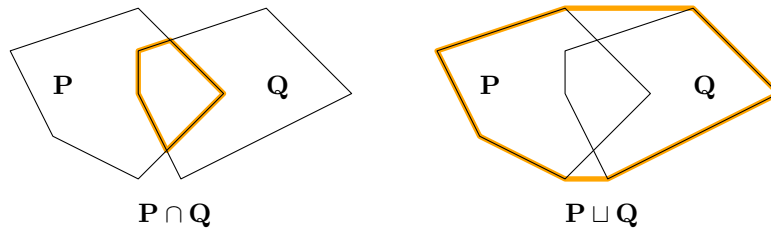


Figure 2.24: Intersection and union of two polytopes. In the case of the union, the convex hull operator  $\sqcup$  is used to stay in the space of convex polytopes. This produces an over approximation:  $\mathbf{P} \cup \mathbf{Q} \subset \mathbf{P} \sqcup \mathbf{Q}$ .

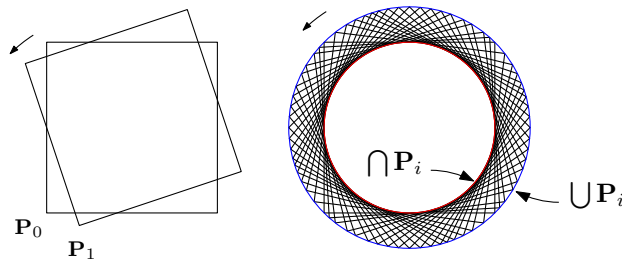


Figure 2.25: A square  $\mathbf{P}_0$  of  $\mathbb{R}^2$  is rotated to form a sequence  $(\mathbf{P}_i)$ . The intersection, respectively the union, of all elements of the infinite sequence is not a polytope but a disk.

An important issue with polytopes is that there is no abstract function  $\alpha$  (see Definition 2.46) between the sets of  $\mathbb{R}^n$  and convex polytopes. The set of polytopes do not have a closure operator (see Definition 2.57). Indeed, let us take  $\mathbf{P}_0 \in \mathbb{P}\mathbb{R}^2$ , a square, and  $\mathbf{P}_i$  the rotation of  $i$  radian of  $\mathbf{P}_0$  (see Figure 2.25). An infinite union of the squares gives a disk which is not a convex polytope:

$$\bigcup_{i=0}^{\infty} \mathbf{P}_i \notin \mathbb{P}\mathbb{R}^2.$$

More generally, the infinite intersection or union of convex polytopes gives the set of closed convex sets. This implies that there is no *best abstraction* of a set using convex polytopes. In other words, we cannot find a unique best polytope that abstracts a set (see Theorem 2.51), *i.e.* there is no  $\alpha$ . Moreover, the set of convex polytopes is also not a complete lattice and do not verify the ACC.

To use convex polytopes, we then need to use a *widening operator* (see Definition 2.60). It will basically give an over-approximation of the convex hull with a limited number of constraints. In this work, we have been using



the *Parma Polyhedra Library*<sup>12</sup> (Bagnara, Hill, and Zaffanella, 2008) to deal with polytopes. This library implements a widening operator.

To conclude, the main drawbacks of convex polytopes compared to boxes is that they are slower to compute, they take more memory, and they do not fulfill the ACC. Moreover, using inclusion functions with polytopes sets is limited for now to linear transformations in available libraries.

### 2.2.2.3 Paving and subpaving

Using a single simple domain such as boxes or convex polytopes to represent sets leads to a large over-approximation arising from the *loss of information*. A straightforward idea is then to use a set of simple domains instead of a single domain to represent sets of  $\mathbb{R}^n$ .

In this section, we will present a particular set of simple domains called subpaving. We will only focus on subpaving of boxes. The subpaving is a tool of the IA community and is barely used by the AI community as they do not form a Galois connection with the set of  $\mathbb{R}^n$  (see Remark 2.87 hereafter).

**Definition 2.86.** A *paving*  $\mathcal{P}$  of  $\mathbb{R}^n$  is a union of non-overlapping boxes with non-zero width of  $\mathbb{I}\mathbb{R}^n$  that covers  $\mathbb{R}^n$ .

A *subpaving* of a box  $[\mathbf{x}] \in \mathbb{I}\mathbb{R}^n$  is a union of non-overlapping subboxes of  $[\mathbf{x}]$  with non-zero width that covers  $[\mathbf{x}]$ .

*Remark 2.87.* There is no Galois connection between the subpavings and the sets of  $\mathbb{R}^n$ . Let us consider an example of a subpaving and a set of  $\mathbb{R}^2$ . We set the order relation  $\subset$  such that for two subpavings  $\mathcal{A}, \mathcal{B}$ ,  $\mathcal{A} \subset \mathcal{B}$  if the set union  $\cup$  of all boxes of  $\mathcal{A}$  is included in the set union of all boxes of  $\mathcal{B}$ . Let us consider now the L shape of Figure 2.26. We want to find a subpaving that minimize the number of boxes but also the loss of information. We can see that there do not exist a unique *best abstraction* as we can choose two minimum different subpavings.

Similarly, the same problem appears if we consider the problem of building the subpaving of a circle.

**Example 2.88.** Subpavings approximate better sets than a single box. In Figure 2.27a,  $[\mathbf{x}_0]$  or  $[\mathbf{x}_1]$  are *outer approximations* of the set  $\mathbb{X}$  (i.e.  $[\mathbf{x}_0] \supseteq \mathbb{X}$  and  $[\mathbf{x}_1] \supseteq \mathbb{X}$ ). We also have that  $[\mathbf{x}_0]$  is a better outer approximation of  $\mathbb{X}$  than  $[\mathbf{x}_1]$  as  $[\mathbf{x}_0] \subset [\mathbf{x}_1]$ . In Figure 2.27b,  $[\mathbf{x}_2]$  is an *inner approximation* of the set  $\mathbb{X}$  (i.e.  $[\mathbf{x}_2] \subseteq \mathbb{X}$ ). Due to the *wrapping effect* (see Remark 2.80), a subpaving can better outer approximate the set  $\mathbb{X}$  than  $[\mathbf{x}_0]$ . This is the case on Figure 2.27c where we have  $\mathbb{X} \subseteq \bigcup_i [\mathbf{x}_i] \subseteq [\mathbf{x}_0]$ . We could have also built a subpaving to under approximate the set  $\mathbb{X}$ .

<sup>12</sup><https://www.bugseng.com/ppl>

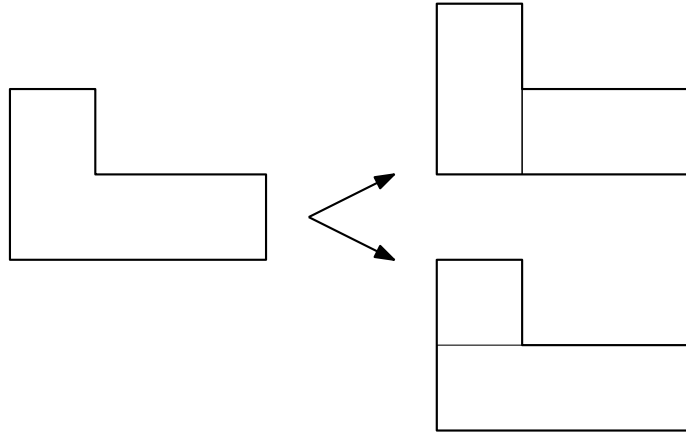


Figure 2.26: On the left an L shape which is a set of  $\mathbb{R}^2$  and on the right, two possible subpavings of the set.

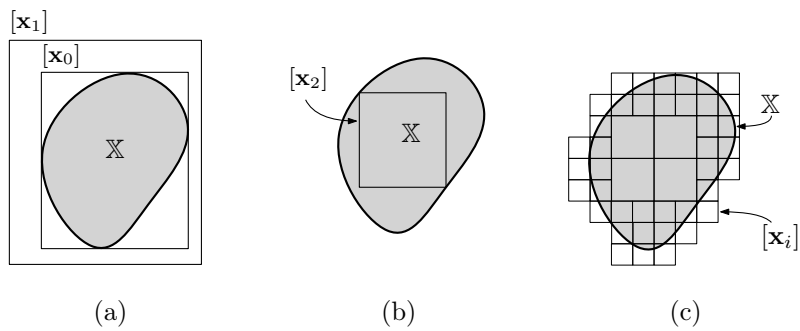


Figure 2.27: Illustration of outer and inner subpavings of a set  $\mathbb{X}$ .

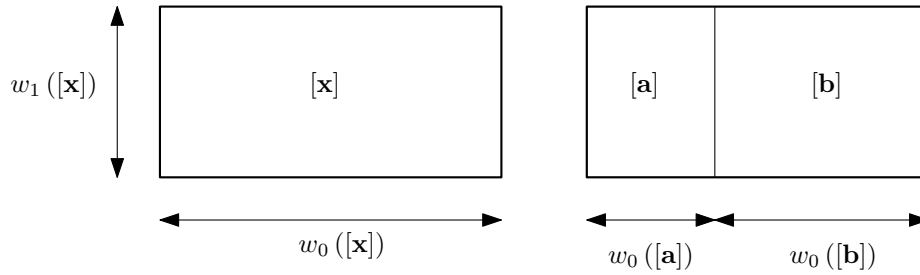


Figure 2.28: Illustration of a bisector in  $\mathbb{R}^2$  with a ratio  $r$ .

A subpaving can be refined using a *bisector*.

**Definition 2.89.** A *bisector* is a function  $\beta: \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n \times \mathbb{R}^n$  such that  $\beta([\mathbf{x}], r) = \{[\mathbf{a}], [\mathbf{b}]\}$  and that verifies (i)  $[\mathbf{a}]$  and  $[\mathbf{b}]$  do not overlap, (ii)  $[\mathbf{x}] = [\mathbf{a}] \cup [\mathbf{b}]$  and (iii)  $w_i([\mathbf{b}]) = r \cdot w_i([\mathbf{x}])$  where  $i$  is the first dimension such that  $\forall k \in \llbracket 0, n \rrbracket, w_i([\mathbf{x}]) \geq w_k([\mathbf{x}])$ .

**Example 2.90.** Let  $[\mathbf{x}] \in \mathbb{R}^2$  and  $w_0([\mathbf{x}]) \geq w_1([\mathbf{x}])$  (see Figure 2.28). We apply the *bisector* to  $[\mathbf{x}]$  with  $r = 0.375$ . We obtain the two boxes  $[\mathbf{a}]$  and  $[\mathbf{b}]$  where  $w_0([\mathbf{b}]) = r \cdot w_0([\mathbf{a}])$ .

*Remark 2.91.* Bisectors and pavings can be generalized to other simple domains, but as stated in the introduction, we will only need in this work to use bisectors applied to boxes. In this document, the  $r$  parameter of the bisector is assumed to be equal to 0.5 unless otherwise stated.

Bisectors can be used recursively to build a subpaving from an initial box  $[\mathbf{x}] \in \mathbb{R}^n$ . A subpaving is called *regular* if each of its boxes is obtained through a finite succession of chosen bisections. In this case, it can be represented as a binary tree where each child is the result of a bisection (see Figure 2.29). We can define the left child and right child as  $\beta([\mathbf{x}]) = \{L[\mathbf{x}], R[\mathbf{x}]\}$  where  $L$  and  $R$  can be viewed as operators from  $\mathbb{R}^n$  to  $\mathbb{R}^n$ .

Additionally, we can define another type of bisection heuristics. This is interesting when the operator is applied recursively. In most cases, it is chosen to bisect at each step the *largest first component* of the box.

*Remark 2.92.* Bisectors are not used, to our knowledge, in the AI community. Indeed, using such a tool implies to define a heuristic which means, most of the time, that there is no Galois connection. Moreover, bisecting produces a combinatory explosion that could additionally explain why this tool is not studied in the AI community.

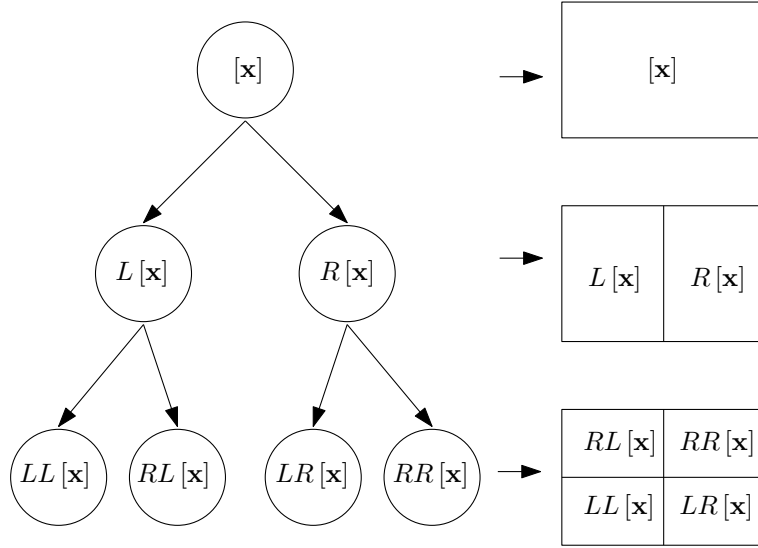


Figure 2.29: Tree representation of a subpaving built with a bisector. The tree has a binary structure.

Abstract domain	Moore family	Loss of information	Computation costs
Polytopes	No	+	+++
Octagons	Yes	++	++
Intervals	Yes	+++	+

Table 2.4: Summary of abstract domains properties.

### 2.2.3 The choice of an abstract domain

Choosing the proper abstract domain for a problem is important to avoid over approximation or convergence issues.

To guarantee the convergence of algorithms, a domain that fulfills the ACC is mandatory unless a widening operator is used. A compromise has to be found between the loss of information due to the abstraction and the computation costs. In this context, using subpaving appears to be an interesting way to reduce the wrapping effect.

The properties of some abstract domains have been summarized in Table 2.4. We have had Octagons that could be a good compromise between polytopes and intervals, but they have not been studied in this work. However, we will see later that in the algorithms of Chapter 3, the use of polytopes will be mandatory to avoid convergence issues.

## 2.3 Constraint programming

Validating that a robot will accomplish its mission correctly can be expressed as constraints on the paths of a dynamical system. From the previous section, we have seen how sets of  $\mathbb{R}^n$ , which were used to abstract paths, can be represented in a computer. We will now see how constraint problems on these sets can be formalized and solved. To this end, we introduce the Constraint Programming framework.

The early history of Constraint Programming (CP) starts in the 1960s. One of the first reference paper dealing with CP was published in 1974 by Montanari (Montanari, 1974) to deal with picture processing. CP has then been the subject of numerous works (Rossi, Van Beek, and Walsh, 2006) and has now a large community.

The main idea of CP is to compute or over-approximate the solutions of complex problems which are modeled using constraints. Although the CP community shares a significant number of ideas with the AI community, their goal does not coincide (Pelleau et al., 2013). Indeed, the CP community focuses on finding the solution (mostly an outer approximation) of problems of the form  $f(x) \geq 0$  whereas the AI community rather looks for the solutions of  $x = f(x)$  problems. In the second case, a fixed point must be reached to conclude unlike the first case where only an outer approximation is required.

In this section, we will introduce the notions of constraint network and fixed points. We will then look at the different ways they can be computed by taking advantage of both CP and AI ideas. Finally, we will deal with some examples to illustrate the approach.

### 2.3.1 Constraint network and fixed points

#### 2.3.1.1 Definition

**Definition 2.93.** A constraint network (CN)  $\mathcal{H}$  is composed of (Russell, Norvig, and Davis, 2010; Rossi, Van Beek, and Walsh, 2006):

- a set of variables  $\mathcal{V} = \{x_1, \dots, x_n\}$ ,
- a set of domains  $\mathcal{D} = \{\mathbb{X}_1, \dots, \mathbb{X}_n\}$  associated to the  $x_i$ ,
- and a set of constraints  $\mathcal{C} = \{c_1, \dots, c_m\}$  that specifies allowable combinations of values. Each constraint  $c \in \mathcal{C}$  is a pair  $c = \langle \sigma, \rho \rangle$  where  $\sigma$ , the constraint scope, is a list of variables, and  $\rho$ , the constraint relation, is a subset of the cartesian product of their domains.

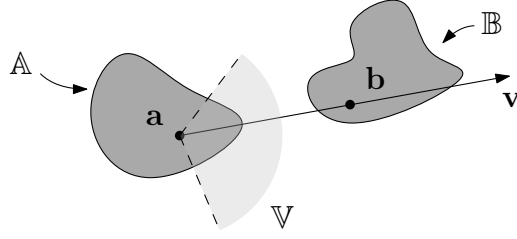


Figure 2.30: The visibility problem of Example 2.95.

Finding the set  $\mathbb{S} \subset \mathcal{D}$  solution of the CN means to solve a *Constraint Satisfaction Problem* (CSP).

*Remark 2.94.* The domains  $\mathcal{D}$  of the CN are usually intervals, boxes (Jaulin et al., 2001), zonotopes (Combastel, 2005), tubes (Drevelle and Bonnifait, 2013; Rohou et al., 2017) or more generally any abstract domains as seen in Section 2.2. We will assume that domains are at least partially ordered sets (see Definition 2.19 on page 24).

The variables could be the paths of  $\mathbb{R}^n$  and the constraints can be, for instance: “all paths starting from  $\mathbb{X}_1$  should not intersect  $\mathbb{X}_2$ ” where  $\mathbb{X}_1$  and  $\mathbb{X}_2$  are sets of  $\mathbb{R}^n$ .

**Example 2.95** (Visibility problem). We consider here a simple example. Let us take two sets  $\mathbb{A}$  and  $\mathbb{B}$  of  $\mathbb{R}^2$  and a set of vectors  $\mathbb{V}$  of  $\mathbb{R}^2$ . The set  $\mathbb{B}$  is said to be visible from  $\mathbb{A}$  if there exists a ray of direction  $\mathbf{v} \in \mathbb{V}$  starting from  $\mathbb{A}$  that intersects  $\mathbb{B}$  (see Figure 2.30). This problem can be modeled using a CN (Guyonneau, Lagrange, and Hardouin, 2013):

- The variables are  $\mathbf{a}$ ,  $\mathbf{b}$  and  $\mathbf{v}$ , they are respectively points of the set  $\mathbb{A}$ ,  $\mathbb{B}$  and  $\mathbb{V}$ ;
- the domains are the sets  $\mathbb{A}$ ,  $\mathbb{B}$  and  $\mathbb{V}$ , *i.e.*  $\mathcal{D} = \{\mathbb{A}, \mathbb{B}, \mathbb{V}\}$ ;
- the constraint is: “there exists a ray of direction  $\mathbf{v}$  starting from  $\mathbf{a}$  that intersects  $\mathbf{b}$ ”.

### 2.3.1.2 Fixed point

To find the solution set  $\mathbb{S}$  of a CN, we will use functions that will modify the domains according to the constraints. To use these functions we first need to introduce the notion of fixed points and associated theorems. Indeed, these tools will give the framework to prove that domains can progressively converge towards the set  $\mathbb{S}$  by applying dedicated functions.

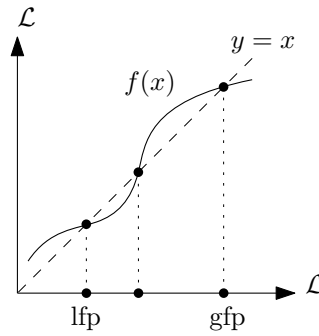


Figure 2.31: Illustration of the Knaster-Tarski theorem. Each  $\bullet$  is a fixed point.

**Definition 2.96.** Let  $f: \mathcal{A} \mapsto \mathcal{A}$  be a function, a *fixed point* is an element  $x \in \mathcal{A}$  such that  $f(x) = x$ .

**Definition 2.97.** Let  $\langle \mathcal{L}, \leq \rangle$  be a lattice and  $f: \mathcal{L} \mapsto \mathcal{L}$  a function.

- The *least fixed point* (*lfp* or *smallest fixed point*) is, if it exists, the unique fixed point that is smaller than each other fixed point.
- The *greatest fixed point* (*gfp*) is, if it exists, the unique fixed point that is greater than each other fixed point.

We denote  $\text{lfp}_{\mathbb{X}}$  the least fixed point that is a subset of  $\mathbb{X}$  and  $\text{gfp}_{\mathbb{X}}$  the greatest fixed point that is a superset of  $\mathbb{X}$ .

**Theorem 2.98** (Knaster-Tarski). (*Tarski, 1955*) Let  $\langle \mathcal{L}, \leq \rangle$  be a complete lattice, let  $f: \mathcal{L} \mapsto \mathcal{L}$  be an increasing function and let  $\mathbb{P}$  be the set of all fixed points of  $f$ . Then the set  $\mathbb{P}$  is not empty and  $\langle \mathbb{P}, \leq \rangle$  is a complete lattice.

*Remark 2.99.* The previous theorem guarantee that there exists a *least fixed point* and a *greatest fixed point* for  $f$  as complete lattices cannot be empty. The result is also true for any decreasing function. Figure 2.31 illustrates the theorem.

Choosing a domain that fulfilled the ACC will allow to build an algorithm that reaches a fixed point in a finite number of steps. Therefore, it will be possible to implement, in a computer, an algorithm that reaches a fixed point.

The following theorem makes the link between a fixed point in the abstract domain and its counterpart in the concrete domain. By verifying the following hypothesis we can compute a fixed point on the abstract domain and then prove that it is at least an over-approximation of the fixed point in the concrete domain.

**Theorem 2.100** (Exact fixed point transfer). *We assume that  $\langle \mathcal{D}, \leq \rangle$  and  $\langle \mathcal{D}^\sharp, \leq^\sharp \rangle$  are complete lattices. We consider a Galois connection  $\langle \mathcal{D}, \leq \rangle \xrightarrow[\alpha]{\gamma} \langle \mathcal{D}^\sharp, \leq^\sharp \rangle$ , two functions  $f: \mathcal{D} \rightarrow \mathcal{D}$  and  $f^\sharp: \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$ , and two elements  $d_0 \in \mathcal{D}$  and  $d_0^\sharp \in \mathcal{D}^\sharp$  such that (Miné, 2013):*

- $f$  is continuous,
- $f^\sharp$  is monotone,
- $\alpha \circ f = f^\sharp \circ \alpha$ ,
- $\alpha(d_0) = d_0^\sharp$ .

We then have:

- both  $f$  and  $f^\sharp$  have a lfp (see Theorem 2.98),
- $\alpha(\text{lfp}_{d_0} f) = \text{lfp}_{d_0^\sharp} f^\sharp$ .

**Example 2.101.** We will illustrate Theorem 2.100 in the case of a *gfp*. Let us take the following Galois connection  $(\mathcal{P}(\mathbb{R}^n), \subseteq) \xrightarrow[\alpha]{\gamma} (\mathbb{IR}^n, \subseteq)$ , and a function  $f$  that contracts sets of  $\mathbb{R}^n$ .<sup>13</sup> We assume that  $f^\sharp$  is the best abstract transformer of  $f$ .  $\mathbb{X}_0 \in \mathcal{P}(\mathbb{R}^n)$  is the initial set. On the top left of Figure 2.32 we can see that the set  $\mathbb{X}_0$  is being contracted by  $f$  until the computation reaches the greatest fixed point as we verify the hypothesis of Theorem 2.98. On the top left, the set  $\mathbb{X}_0$  is abstracted by a box  $[\mathbb{Y}_0] = \alpha(\mathbb{X}_0)$ . The best abstract transformer  $f^\sharp$  is then applied several times until a *gfp* is reached. On the bottom of Figure 2.32, we have shown the evolution of the sets. We obtain at the end:  $\alpha(\text{gfp}_{\mathbb{X}_0} f) = \text{gfp}_{[\mathbb{X}_0]} f^\sharp$  which corresponds to an exact fixed point transfer. Due to the abstraction, we can notice that we have  $\text{gfp}_{\mathbb{X}_0} f \subseteq \gamma(\text{gfp}_{[\mathbb{X}_0]} f^\sharp)$ .

*Remark 2.102.* In most cases, we only have  $\alpha \circ f \subseteq f^\sharp \circ \alpha$ . This is the case when we do not have the best abstract transformer for  $f$  but only a sound abstract transformer (see Definition 2.54). This can still be the case with the best abstract transformer if we do not have completeness, *i.e.* if we do not have  $\alpha \circ f = \alpha \circ f \circ (\gamma \circ \alpha)$  but only  $\alpha \circ f \subseteq \alpha \circ f \circ (\gamma \circ \alpha)$ .

In these cases, the result of Theorem 2.100 becomes  $\alpha(\text{lfp}_{d_0} f) \subseteq \text{lfp}_{d_0^\sharp} f^\sharp$ .

*Remark 2.103.* When there is no Galois connection, we can choose a monotone function  $\varphi$  that plays the role of the abstract function  $\alpha$ . We have then to ensure that  $\varphi \circ f \subseteq f^\sharp \circ \varphi$  and  $\varphi(d_0) \subseteq d_0^\sharp$  to prove that  $\varphi(\text{lfp}_{d_0} f) \subseteq \text{lfp}_{d_0^\sharp} f^\sharp$ .

<sup>13</sup>Idem. footnote 9 on page 40.



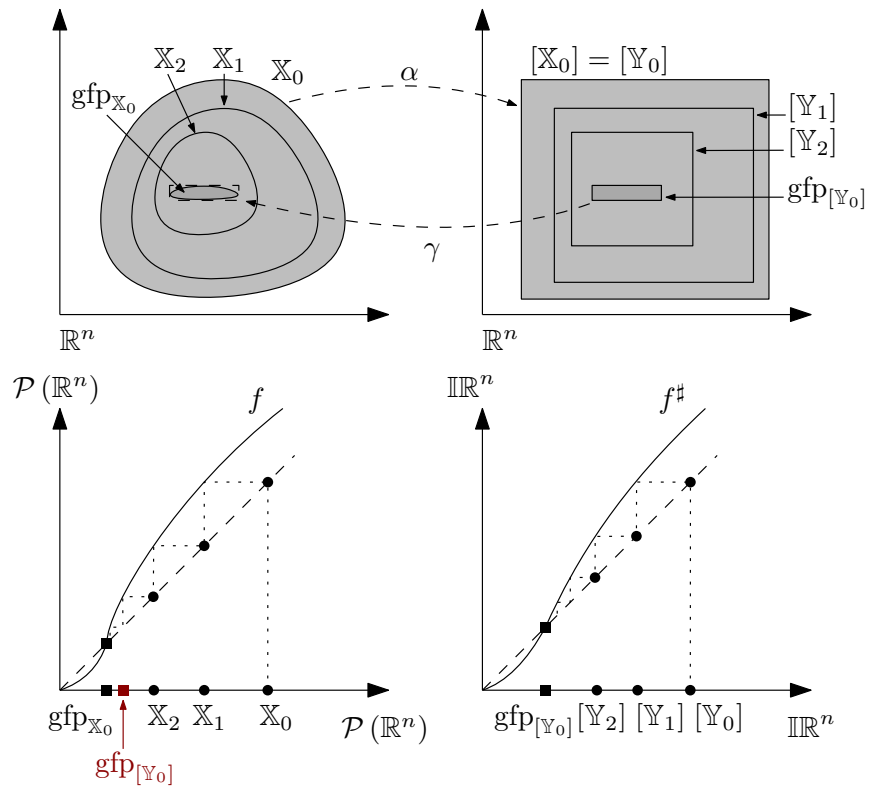


Figure 2.32: Illustration of the exact fixed point transfer with the Galois connection  $(\mathcal{P}(\mathbb{R}^n), \subseteq) \xrightleftharpoons[\alpha]{\gamma} (\mathbb{IR}^n, \subseteq)$  and a function  $f$ , a contractor, and its best abstract transformer  $f^\#$ .

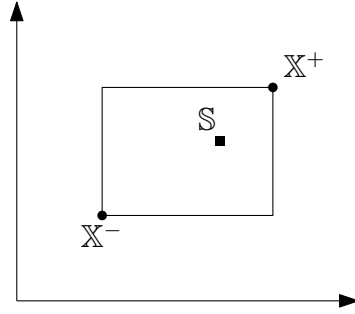


Figure 2.33: Bracketing the solution set  $\mathbb{S}$  of a CN with an interval  $[\mathbb{X}^-, \mathbb{X}^+]$ .

### 2.3.2 Bracketing the solution set of a Constraint Network

We recall that we want to approximate the solution set  $\mathbb{S}$  of a CN. It is important to note that, most of the time, the set  $\mathbb{S}$  cannot be represented in a computer. Therefore, we will abstract  $\mathbb{S}$  by an interval of sets that brackets  $\mathbb{S}$  (see Figure 2.33).

**Definition 2.104.** A set  $\mathbb{X}^+$  is an outer approximation of the solution set  $\mathbb{S}$  of CN  $\mathcal{H}$  if  $\mathbb{S} \subseteq \mathbb{X}^+$ . A set  $\mathbb{X}^-$  is an inner approximation if  $\mathbb{X}^- \subseteq \mathbb{S}$ . Bracketing the solution of  $\mathcal{H}$  consists in finding two sets  $\mathbb{X}^+$  and  $\mathbb{X}^-$  such that  $\mathbb{X}^- \subseteq \mathbb{S} \subseteq \mathbb{X}^+$ . We have  $\mathbb{S} \in [\mathbb{X}^-, \mathbb{X}^+]$ .

To obtain an inner  $\mathbb{X}^-$  or an outer  $\mathbb{X}^+$  approximation of the CSP from an initial state of the domain  $\mathbb{X}_0$ , we can build several functions  $f: \mathcal{D} \mapsto \mathcal{D}$  with the following four *strategies* which will be studied in detail, in the next sections:

1. remove non-solutions without removing any solutions of  $\mathbb{X}_0$  to obtain a  $\mathbb{X}^+$ ,
2. remove all non-solutions of  $\mathbb{X}_0$  to obtain a  $\mathbb{X}^-$ ,
3. add all solutions to  $\mathbb{X}_0$  to obtain a  $\mathbb{X}^+$ ,
4. add solutions without adding any non-solutions to  $\mathbb{X}_0$  to obtain a  $\mathbb{X}^-$ .

Removing solutions is an operation that *contracts*, i.e.  $f(x) \leq x$ , the domains, while adding solutions *inflates*, i.e.  $f(x) \geq x$ , the domains. We give hereafter a definition of the two operators.

**Definition 2.105.** A *contractor*  $\mathcal{C}$  (Chabert and Jaulin, 2009) is an operator  $\text{MIR}^n \mapsto \text{MIR}^n$ , such that for all  $\mathbf{x}, \mathbf{y} \in \text{MIR}^n$ ,

- $\mathcal{C}(\mathbf{x}) \subseteq \mathbf{x}$  (contractance)
- $\mathbf{x} \subset \mathbf{y} \implies \mathcal{C}(\mathbf{x}) \subset \mathcal{C}(\mathbf{y})$  (monotonicity)

**Definition 2.106.** Similarly to the contractor, an *inflator*  $\mathcal{I}$  is defined as an operator  $\text{MIR}^n \mapsto \text{MIR}^n$ , such that for all  $\mathbf{x}, \mathbf{y} \in \text{MIR}^n$ ,

- $\mathcal{I}(\mathbf{x}) \supseteq \mathbf{x}$  (inflation)
- $\mathbf{x} \subset \mathbf{y} \implies \mathcal{I}(\mathbf{x}) \subset \mathcal{I}(\mathbf{y})$  (monotonicity)

The function  $f$  can be applied several times to the domain until a fixed point is reached, *i.e.* when no more solutions can be added or non-solutions removed. To ensure that the function will reach a fixed point, we recall that we have to verify that:  $f$  is an increasing function, *i.e.*  $x \leq y \implies f(x) \leq f(y)$ , and that  $\langle \mathcal{D}, \leq \rangle$  is a lattice (see Theorem 2.98).

*Remark 2.107.* In practice, *contractors* or *inflators* will be built on abstract domains to be implemented in a computer. For instance, the strategy 1: “remove non-solutions without removing any solutions of  $\mathbb{X}_0$ ” is commonly used in the IA community. The function will then reach a fixed-point, which is an over or an under approximation of  $\mathbb{S}$ , as proved by the fixed point transfer theorem (see Theorem 2.100).

We will now focus successively on the computation of the outer approximation and then on the computation of the inner approximation.

### 2.3.2.1 Outer approximation

To find an outer approximation, we need to obtain a set  $\mathbb{X}^+$  such that  $\mathbb{S} \subseteq \mathbb{X}^+$ . It means that we may have in  $\mathbb{X}^+$  non solutions together with all solutions of the CSP.

To this end, there are two different approaches which depend on the initial set  $\mathbb{X}_0$  we have, and on the difficulty of finding a function  $f$ :

- if  $\mathbb{X}_0$  is a superset of  $\mathbb{S}$  ( $\mathbb{S} \subseteq \mathbb{X}_0$ ) then we can use a contractor approach and build a function  $f$  which verifies *strategy* (1),
- if  $\mathbb{X}_0$  is a subset of  $\mathbb{S}$  ( $\mathbb{X}_0 \subseteq \mathbb{S}$ ) then we can use an inflator approach and build a function  $f$  which verifies *strategy* (3).

Figure 2.34 illustrates the contraction approach where  $\mathbb{S} \subseteq \mathbb{X}_0$ . If  $f$  is applied several times, the contracted domain will converge to the greatest fixed point  $f^\infty(\mathbb{X}_0) = f \circ f \circ \dots \circ f(\mathbb{X}_0) = \mathbb{X}_\infty$  which is still an outer approximation of  $\mathbb{S}$ . The accuracy of the outer approximation will depend on the efficiency of

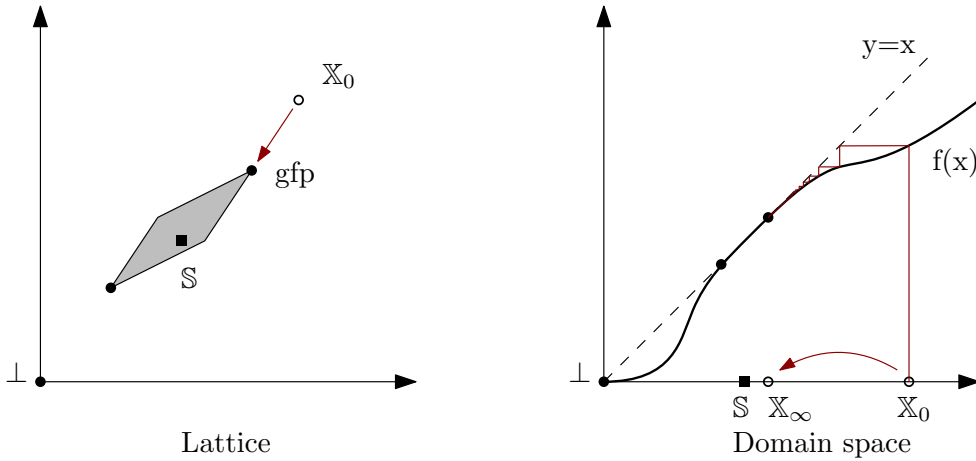


Figure 2.34: A function that *contracts* domains to compute an outer approximation of  $\mathbb{S}$ . The grey polygon is the set of all fixed points for the function.

$f$  to remove all non-solutions. We do not necessarily need to reach the fixed point to obtain an outer approximation as  $f^k(\mathbb{X}_0)$  is always an outer approximation. When a fixed point is reached, the domain will not be contracted anymore by applying  $f$ , *i.e.*  $f(\mathbb{X}_\infty) = \mathbb{X}_\infty$ .

Figure 2.35 illustrates the inflation approach with  $\mathbb{X}_0 \subseteq \mathbb{S}$ . Contrary to the contraction case, the fixed point of  $f$  has to be reached to obtain  $\mathbb{S} \subseteq \mathbb{X}^+$  as  $f$  is designed to validate this property only at the fixed point. After reaching the fixed point, which is the least fixed point in Figure 2.35, we can verify that  $\mathbb{X}_\infty \supseteq \mathbb{S}$ .

Depending on the problem it is easier to have  $\mathbb{X}_0 \subset \mathbb{S}$  and to use an inflator approach, or  $\mathbb{X}_0 \supset \mathbb{S}$  and to use a contractor approach.

### 2.3.2.2 Inner approximation

To find an inner approximation, the same approach as the outer approximation can be applied but with the use of *strategy* (2) and (4). However, finding an inner approximation requires most of the time that the solution set has a non-zero volume (Lebesgue measure). For instance if the solution set of a CN is the limit cycle of the Van Der Pol system, it is possible to obtain an outer approximation but it will not be possible to obtain, numerically with a computer, an inner approximation. Moreover, to our knowledge, there is no analytical expression of the limit cycle.

We have similarly:

- if  $\mathbb{X}_0$  is a superset of  $\mathbb{S}$  ( $\mathbb{S} \subseteq \mathbb{X}_0$ ) then we can use a contractor approach

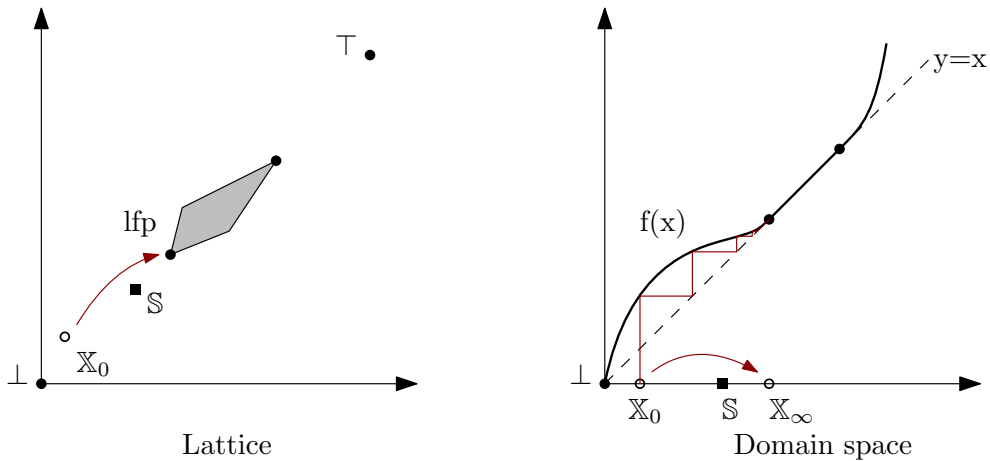


Figure 2.35: A function that *inflates* domains to compute an outer approximation of  $\mathbb{S}$ . The grey polygon is the set of all fixed points for the function.

and build a function  $f$  with the property (2) (see Figure 2.36),

- if  $\mathbb{X}_0$  is a subset of  $\mathbb{S}$  ( $\mathbb{X}_0 \subseteq \mathbb{S}$ ) then we can use an inflator approach and build a function  $f$  with the property (4) (see Figure 2.37).

*Remark 2.108.* In some cases, it is interesting to consider the complementary CN  $\overline{\mathcal{H}}$ . Indeed, a function  $f$  that contracts or inflates an initial set  $\overline{\mathbb{X}_0}$  might be easier to implement with the complementary form of the CN. This idea will be used in Chapter 3.

### 2.3.2.3 Example

Let us consider again the visibility problem of Example 2.95. We want to find an over approximation of the set  $\mathbb{S} \subset \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$  solution of the constraint network. The constraint can be rewritten as:

$$\forall \mathbf{a} \in \mathbb{A}, \forall \mathbf{b} \in \mathbb{B}, \exists \mathbf{v} \in \mathbb{V}, \exists t \geq 0, \mathbf{a} + t\mathbf{v} = \mathbf{b}$$

which can be divided into two constraints with the intermediate variable  $\mathbf{c}$  and the projection of  $\mathbf{v}$  and  $\mathbf{c}$ :

$$\begin{cases} \mathbf{c} = \mathbf{b} - \mathbf{a} \\ t = c_1/v_1 \\ t = c_2/v_2 \end{cases}$$

We chose to use intervals to abstract the sets, and we will assume that we are in the case where  $\mathbb{S} \subset \mathbb{X}_0$ . To obtain an outer approximation of  $\mathbb{S}$ ,

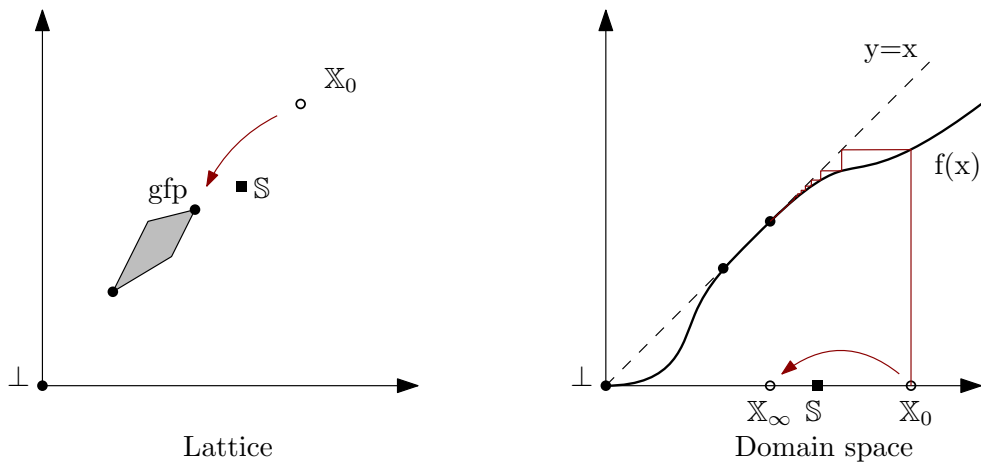


Figure 2.36: A function that *contracts* domains to compute an inner approximation of  $S$ . The grey polygon is the set of all fixed points for the function.

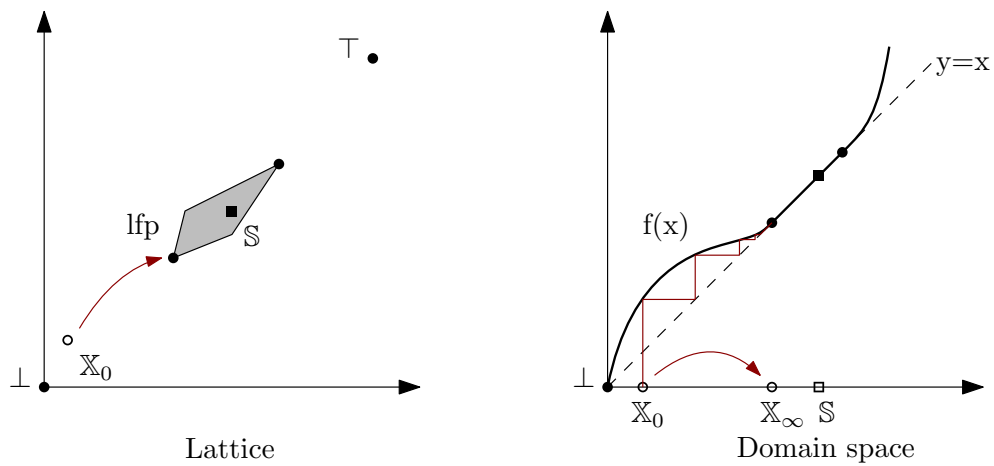


Figure 2.37: A function that *inflates* domains to compute an inner approximation of  $S$ . The grey polygon is the set of all fixed points for the function.

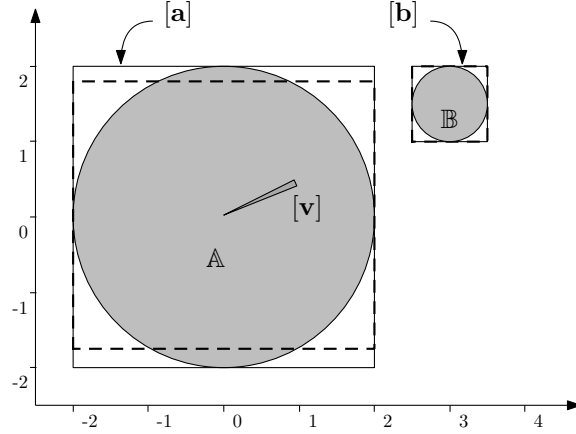


Figure 2.38: Example of the  $\mathcal{C}_{\text{visibility}}$  contractor. The sets  $\mathbb{A}$  and  $\mathbb{B}$  are abstracted using a single box, respectively  $[\mathbf{a}]$  and  $[\mathbf{b}]$ , with the convex hull operator. The result of the contractor is shown in dashed. The gray cone represents the union of the directions of  $[\mathbf{v}]$ .

we define  $[\mathbf{a}]$ ,  $[\mathbf{b}]$  and  $[\mathbf{v}]$ , respectively the box hull of  $\mathbb{A}$ ,  $\mathbb{B}$  and  $\mathbb{V}$ . We also define the contractor  $\mathcal{C}_{\text{visibility}}$  as:

$$\begin{pmatrix} [\mathbf{a}] \\ [\mathbf{b}] \\ [\mathbf{c}] \\ [t] \\ [\mathbf{v}] \end{pmatrix} \mapsto_{\mathcal{C}_{\text{visibility}}} \begin{pmatrix} [\mathbf{a}] \cap ([\mathbf{b}] - [\mathbf{c}]) \\ [\mathbf{b}] \cap ([\mathbf{c}] + [\mathbf{a}]) \\ [\mathbf{c}] \cap ([\mathbf{b}] - [\mathbf{a}]) \cap ([t] \cdot [\mathbf{v}]) \\ [t] \cap ([c_1] / [v_1]) \cap ([c_2] / [v_2]) \\ [\mathbf{v}] \cap ([\mathbf{c}] / [t]) \end{pmatrix}.$$

This contractor only removes non solutions of the domain. By applying the contractor up to the fixed point, we obtain an outer approximation of  $\mathbb{S}$ .

Let  $\mathbb{A}$  be the circle centered in  $(0, 0)^T$  of radius 2,  $\mathbb{B}$  the circle centered in  $(3, 1.5)^T$  of radius 0.5 and  $\mathbb{V} = \{1\} \times [0.4, 0.5]$ . We have  $[\mathbf{a}] = [-2, 2] \times [-2, 2]$ ,  $[\mathbf{b}] = [2.5, 3.5] \times [1, 2]$  and  $[\mathbf{v}] = \{1\} \times [0.4, 0.5]$ . We set  $[t] = [0, \infty]$  and  $[\mathbf{c}] = [-\infty, \infty]$ . Note that we use the box hull operator to abstract the sets of  $\mathbb{R}^2$  to the interval domain.

By applying the contractor  $\mathcal{C}_{\text{visibility}}$  up to the fixed point, we obtain  $[\mathbf{a}] = [-2, 2] \times [-1.75, 1.8]$ . Figure 2.38 shows the contracted boxes (dashed).

Due to the *loss of information* of the boxes, we obtain a large over-approximation of  $\mathbb{S}$ . To improve the result, we can use subpavings as domain instead of boxes. To over-approximate the set  $\mathbb{A}$  and  $\mathbb{B}$  we build two subpavings that over approximate the sets instead of using the convex hull operator.

For simplicity, we will use in this example a simple subpaving  $\mathcal{A}$  for  $\mathbb{A}$  composed of four boxes of same size, and we will keep one box for  $\mathbb{B}$ . We can

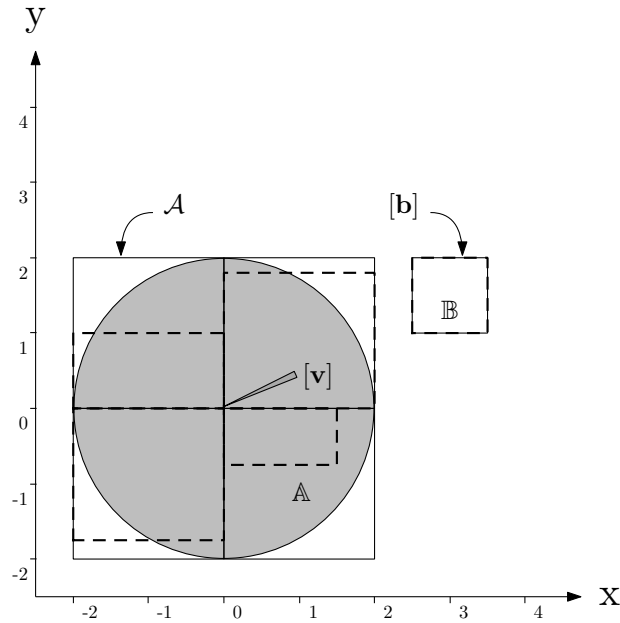


Figure 2.39: Example of the  $\mathcal{C}_{\text{visibility}}$  contractor with a subpaving. This example is identical to Figure 2.38 except that the set  $\mathbb{A}$  is now abstracted using a subpaving  $\mathcal{A}$ . The result is shown in dashed.

see the result in Figure 2.39 after applying the contractions. The obtained set is more accurate than with a single box.

We can note that we have chosen an *ad hoc* function to abstract the set of  $\mathbb{R}^2$  into a subpaving as there is no Galois connection (see Remark 2.87 on page 53). The function works the following way: we use the convex hull of  $\mathbb{A}$  and we apply a bisector  $\beta$  to this hull until we obtain four boxes.

### 2.3.3 Example of algorithms

In this section, we will look at two examples of algorithms that illustrate the CP approach. The ideas on which these algorithms rely, will also be useful to the next chapters. The first one concerns the way a subpaving can be built from a set of constraints and the second one concerns the use of datasets with contractor techniques.

*Remark 2.109* (Color convention). Before we start looking at the examples, we will set a color convention which will be valid for all the following figures of this work.

Let us take back Example 2.88 and let us use the same subpaving to over and under approximate the set  $\mathbb{X}$ . We obtain two sets  $\mathbb{X}^+$  and  $\mathbb{X}^-$  such that



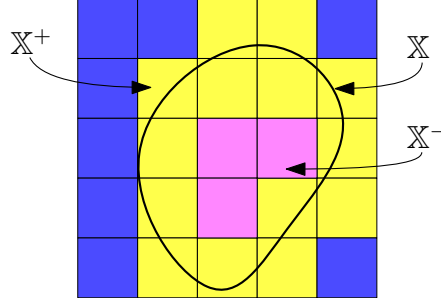


Figure 2.40: Outer and inner subpaving of a set  $\mathbb{X}$  with color convention: magenta inside, blue outside and yellow for the undetermined boxes.

$\mathbb{X}^- \subset \mathbb{X} \subset \mathbb{X}^+$  (see Figure 2.40).

The color convention is the following:  $\mathbb{X}^-$  is composed of the union of all magenta boxes (  $\blacksquare$  ) and  $\mathbb{X}^+$  is composed of all magenta and yellow boxes (  $\blacksquare \cup \blacksquare$  ).

To summarize (with an island analogy)

- Blue boxes  $\blacksquare$  are completely outside the set ( $[\mathbf{x}_i] \cap \mathbb{X} = \emptyset$ ) – *the sea*;
- Magenta boxes  $\blacksquare$  are completely inside the set ( $[\mathbf{x}_i] \cap \mathbb{X} = [\mathbf{x}_i]$ ) – *the ground*;
- Yellow boxes  $\blacksquare$  are undetermined – *the sand*.

### 2.3.3.1 CSP and Set inversion problem

When solving a CSP, we have a set of inequalities of the form  $\mathbf{f}(\mathbf{x}) \geq 0$  which requires to find the  $\mathbf{x}$  solutions. However, we do not have, most of the time, an explicit form of  $\mathbf{f}^{-1}$  which would have allowed to solve the problem analytically. Therefore, we will introduce here the *set inversion problem* and an algorithm to solve this problem.

**Definition 2.110.** Let  $\mathbf{f}$  be a function from  $\mathbb{R}^n$  to  $\mathbb{R}^m$  and  $\mathbb{Y}$  a subset of  $\mathbb{R}^m$ . Set inversion is the characterization of

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\} = \mathbf{f}^{-1}(\mathbb{Y}).$$

The *SIVIA* (Set Inverter Via Interval Analysis) algorithm (Jaulin et al., 2001) is design to build two regular subpavings  $\mathcal{L}^{\text{outer}}$  and  $\mathcal{L}^{\text{inner}}$  such that

$$\bigcup \mathcal{L}^- \subset \mathbb{X} \subset \bigcup \mathcal{L}^+$$

where  $\bigcup$  is the union (not convex hull) of all boxes of the subpaving. The algorithm uses as input an inclusion function  $[\mathbf{f}] : \mathbb{R}^n \rightarrow \mathbb{R}^m$  of  $\mathbf{f}$ , an initial box  $[\mathbf{x}]$ , a paving accuracy  $\epsilon \in \mathbb{R}$  and a set  $\mathbb{Y}$  that can be for instance abstracted by a box (see Algorithm 1).

---

**Algorithm 1:** SIVIA algorithm.

---


**Input:**  $\mathbf{f}, \mathbb{Y}, [\mathbf{x}], \epsilon$   
**Output:**  $\mathcal{L}^-, \mathcal{L}^+$

```

1  $\mathcal{L} = \{[\mathbf{x}]\}, \mathcal{L}^- = \emptyset, \mathcal{L}^+ = \emptyset;$ 
2 while  $\mathcal{L} \neq \emptyset$  do
3   pop  $\mathcal{L}$  into  $[\mathbf{x}];$ 
4   if  $[\mathbf{f}]([\mathbf{x}]) \subset \mathbb{Y}$  then
5      $\mathcal{L}^- \leftarrow \mathcal{L}^- \cup [\mathbf{x}];$ 
6      $\mathcal{L}^+ \leftarrow \mathcal{L}^+ \cup [\mathbf{x}];$ 
7   else if  $[\mathbf{f}]([\mathbf{x}]) \cap \mathbb{Y} \neq \emptyset$  then
8     if  $w([\mathbf{x}]) < \epsilon$  then
9        $\mathcal{L}^+ \leftarrow \mathcal{L}^+ \cup [\mathbf{x}];$ 
10    else
11       $\mathcal{L} \leftarrow \mathcal{L} \cup L[\mathbf{x}] \cup R[\mathbf{x}];$ 
12    end
13  end
14 end

```

---

*Remark 2.111.* When the constraints of a CN are inequalities, the set  $\mathbb{Y}$  is the non-negative orthant of  $\mathbb{R}^m$ . Figure 2.41 illustrates the algorithm in the case where a box is finally bisected. If the set  $\mathbb{X}$  has no volume, as in the case of the limit cycle of the Van Der Pol system in  $\mathbb{R}^2$  (see Example 2.25), no inner approximation will be obtained (*i.e.*  boxes).

**Example 2.112.** We consider the pendulum Example 2.39 where we found  $V(\mathbf{x}) = a(1 - \cos x_1) + \frac{1}{2}x_2^2$ . We can bracket the level-set  $V(\mathbf{x}) \leq l$  with the SIVIA algorithm. For  $a = 1$  and  $l = 2$ , we obtain Figure 2.42. We have also applied a separator technique (Jaulin and Desrochers, 2014), to obtain this figure. This allows to reduce the number of bisections required. The main idea is to contract boxes while bisecting. This is why we do not have a regular paving here.

**Example 2.113.** In this second example, we apply the SIVIA algorithm to the example of Section 2.3.2.3 along with the visibility contractor. We use the SIVIA algorithm to build a subpaving where the constraint is “the box should

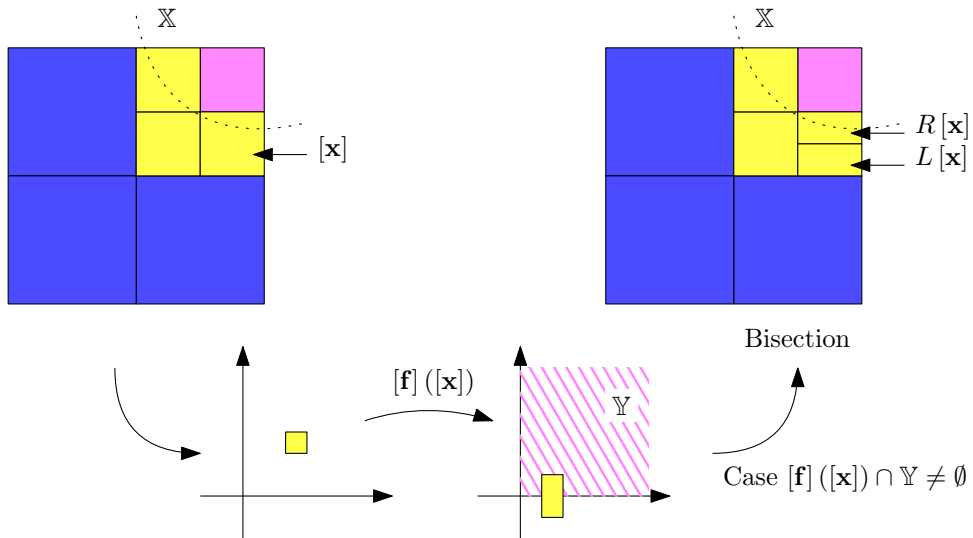


Figure 2.41: Example of the bisection with the SIVIA algorithm in the case where  $[f]([x]) \cap Y \neq \emptyset$ . On the left top: the initial subpaving. On the right top: the subpaving after bisection.

belong to  $\mathbb{A} = \{\mathbf{x} \mid x_1^2 + x_2^2 \leq 2^2\}$ . We simultaneously apply the visibility contractor to each box of the subpaving to fulfill the visibility constraint. Figure 2.43 shows, projected in the space of  $\mathbb{A}$ , the outer subpaving  $\mathcal{L}_{\mathbb{X}^+}$  in yellow.

We can notice that it is more efficient to contract boxes using the visibility constraint than to add the visibility constraint in the SIVIA algorithm. Indeed, the SIVIA algorithm only checks if a box validate the constraints and otherwise may apply a bisection. Using contractors helps to reduce the size of the boxes before bisection (Chabert and Jaulin, 2009).

We can also see that even with a simple example, it is difficult to build dedicated contractors on sets of  $\mathbb{R}^n$  to deal with simple paths (in our case straight lines).

*Remark 2.114.* The SIVIA algorithm can be viewed as a function  $\varphi$  that from a set of constraints associate a subpaving. In the previous example, a set of  $\mathbb{R}^n$  is abstracted by a set of constraints which are abstracted by a subpaving.

### 2.3.3.2 Dealing with datasets

This section deals with the issue of working with datasets such as marine currents. The idea is to show how a contractor approach can be used to efficiently evaluate the boundaries of a large dataset for any given range.

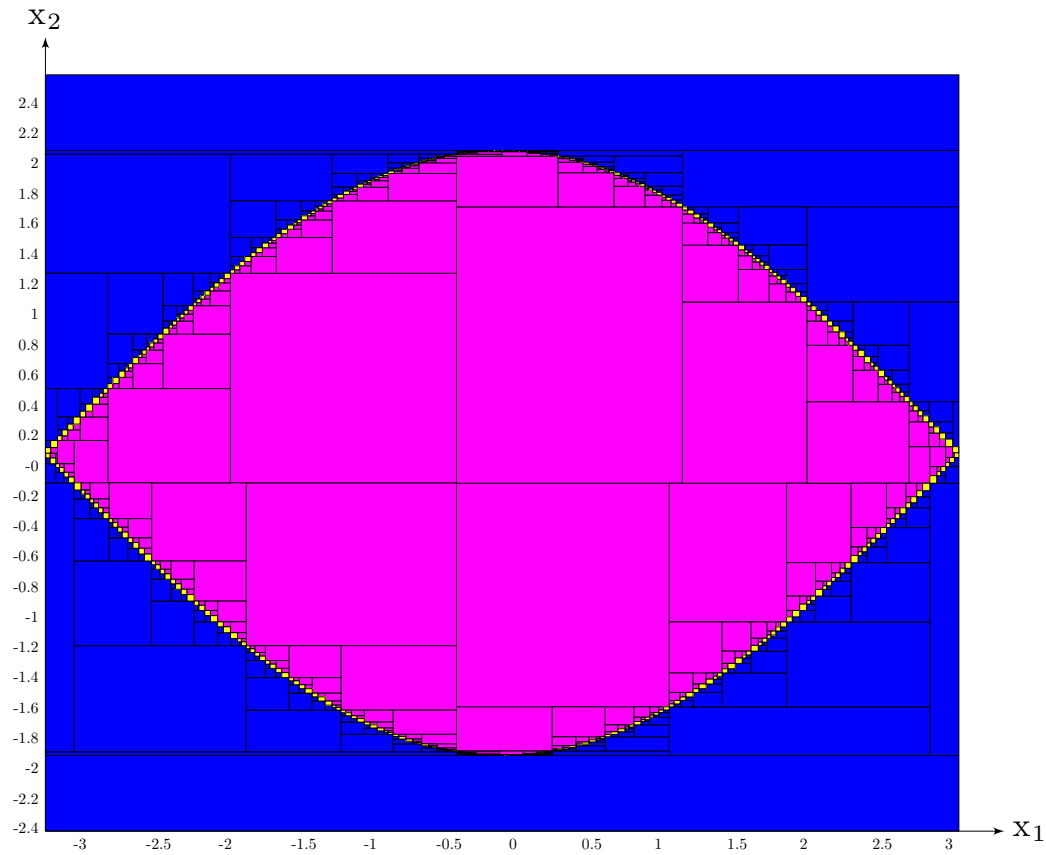


Figure 2.42: Subpaving associated to the level-set  $l = 2$  of the pendulum example using the SIVIA algorithm.

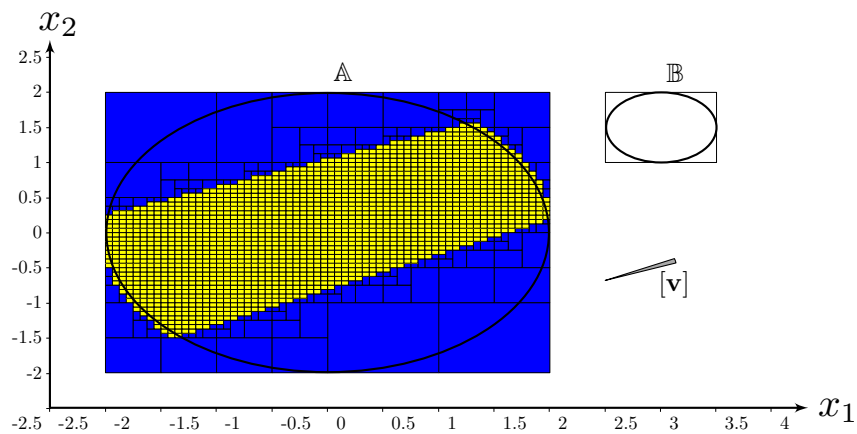


Figure 2.43: Set characterization with the SIVIA algorithm of the visibility problem.  $\mathbb{A}$  and  $\mathbb{B}$  are the two circles and the yellow subpaving corresponds to the position in  $\mathbb{A}$  from where the bounding box of  $\mathbb{B}$  is visible.

**Definition 2.115.** (Jaulin, 2006) A *staircase function*  $\phi$  associated to a paving  $\mathcal{Q}$  of  $\mathbb{R}^n$  is a function from  $\mathbb{R}^n$  to  $\mathbb{R}^m$  which has a constant value for each box of  $\mathcal{Q}$ .

An *interval staircase function*  $\Phi = [\phi^-, \phi^+]$  is a pair of two staircase functions such that  $\phi^- \leq \phi^+$  (see Figure 2.44).

A function  $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$  is said to belong to the interval staircase function  $\Phi$  if

$$\forall [\mathbf{x}] \in \mathcal{Q}, \forall \mathbf{x} \in [\mathbf{x}], f(\mathbf{x}) \in [\phi^-([\mathbf{x}]), \phi^+([\mathbf{x}])].$$

*Remark 2.116.* The definition of interval staircase functions relies on the property that the set of all staircase functions equipped with the natural partial order is a complete lattice:  $\phi^- \leq \phi^+ \Leftrightarrow \forall [\mathbf{x}] \in \mathcal{Q}, \phi^-([\mathbf{x}]) \leq \phi^+([\mathbf{x}])$ .

Interval staircase functions can enclose datasets such as current velocities or bathymetry. Such data are mainly available on regular grid pattern that makes interval staircase functions well suited to deal with such data.

**Proposition 2.117** (*Dataset contractor*). *Given an interval staircase function  $[\Phi]$  from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ , associated to a paving  $\mathcal{Q}$  of  $\mathbb{R}^n$ , and a box  $[\mathbf{x}]$  of  $\mathbb{R}^{n \times m}$ , the dataset contractor  $\mathcal{C}_{dataset}$  is the operator defined by:*

$$[\mathbf{x}] \xrightarrow{\mathcal{C}_{dataset}} [\{\mathbf{x} \in [\mathbf{x}] \mid proj_{\mathbb{R}^m}(\mathbf{x}) \in \Phi(proj_{\mathbb{R}^n}(\mathbf{x}))\}]$$

*Remark 2.118.* The use of a regular paving for  $\mathbb{R}^n$  allows to use an efficient tree structure. An illustration of the contractor is given Figure 2.44. In practice, one of the critical point is to design correctly the interval staircase function when, in particular, only discrete data are given. A correct interpolation must be undertaken to guarantee that no solution will be lost.

**Example 2.119.** Ocean currents are given as a discrete data grid where, at each geographical position ( $\mathbb{R}^2$ ), a north and east velocity of the water ( $\mathbb{R}^2$ ) are associated. To build an interval staircase function, we have to choose an interpolation method between points to guarantee that the function encloses correctly the currents (see Figure 2.45). The smaller the boxes of the paving are, the more accurate the interval staircase function will be. A special case must be taken into account when there is no data available for a box of the paving: the value of the interval staircase function will be set to a value that is relevant for the model. It could be either  $[-\infty, +\infty]^m$ , or the boundaries of maximum possible values, or  $\mathbf{0}^m$ . In the case of ocean currents, we can choose  $\mathbf{0}^2$  when the box corresponds to a position on the land.

We give here a simple implementation of the contractor as a recursive algorithm which uses the tree structure of the dataset (see the example of a

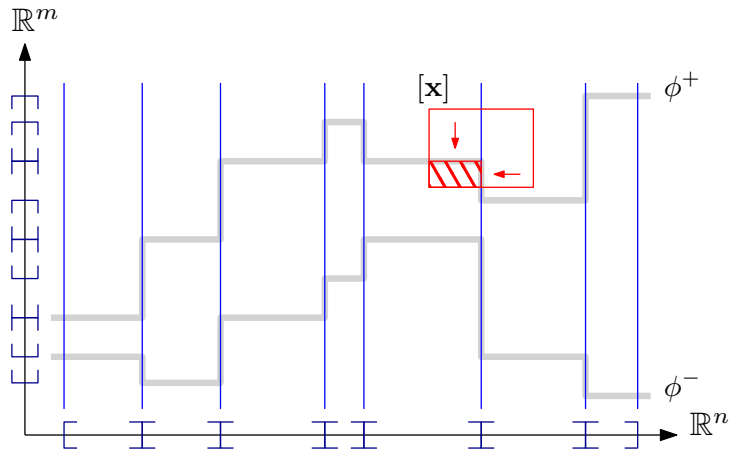


Figure 2.44: Illustration of the staircase contractor for a box  $[x]$  in  $\mathbb{R}^{n \times m}$ . An interval staircase function  $\Phi$ , in gray, has been built on a paving of  $\mathbb{R}^n$ .

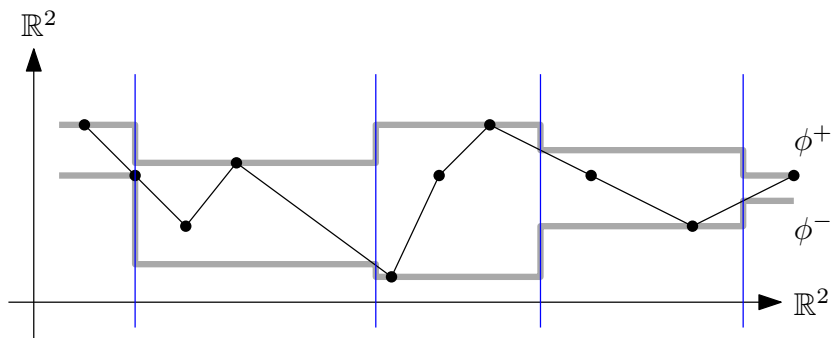


Figure 2.45: Illustration of the interpolation of a grid of current.  $\bullet$  are discrete value of currents at a given position,  $-$  is a chosen interpolation of data which is used to build an interval staircase function  $\Phi$ .

tree in Figure 2.29 on page 56). For each node  $\mathcal{N}$  of the tree, we can associate a box  $[\mathbf{y}]$  for the projection of  $\Phi$  on  $\mathbb{R}^n$ , denoted  $\mathcal{N}_{[\mathbf{y}]}$ , and an associated box  $[\mathbf{v}]$  for the projection on  $\mathbb{R}^m$ , the data, denoted  $\mathcal{N}_{[\mathbf{v}]}$ . We have therefore  $[\mathbf{x}] = [\mathbf{y}] \times [\mathbf{v}]$ . Algorithm 2 shows how to compute the contraction.

---

**Algorithm 2:** Dataset contractor algorithm ( $\mathcal{C}_{\text{dataset}}$ ).

---

**Input:**  $\mathcal{N}, [\mathbf{y}], [\mathbf{v}]$   
**Output:**  $[\mathbf{y}], [\mathbf{v}]$

- 1  $[\mathbf{y}] \leftarrow [\mathbf{y}] \cap \mathcal{N}_{[\mathbf{y}]}$ ;
- 2  $[\mathbf{v}] \leftarrow [\mathbf{v}] \cap \mathcal{N}_{[\mathbf{v}]}$ ;
- 3 **if**  $[\mathbf{v}]$  and  $[\mathbf{y}]$  are not empty **then**
- 4      $[\mathbf{y}_R], [\mathbf{v}_R] = \mathcal{C}_{\text{dataset}}(R(\mathcal{N}), [\mathbf{y}], [\mathbf{v}])$ ;
- 5      $[\mathbf{y}_L], [\mathbf{v}_L] = \mathcal{C}_{\text{dataset}}(L(\mathcal{N}), [\mathbf{y}], [\mathbf{v}])$ ;
- 6      $[\mathbf{y}] \leftarrow [\mathbf{y}_R] \cup [\mathbf{y}_L]$ ;
- 7      $[\mathbf{v}] \leftarrow [\mathbf{v}_R] \cup [\mathbf{v}_L]$ ;

---

**Example 2.120.** To illustrate Remark 2.119, let us take a simple example of dimension  $1 \times 1$  represented in Figure 2.46 of an interval staircase function associated to data and the associated tree structure built on the subpaving.

Let us assume we have an initial box  $[\mathbf{x}] = [6, 11] \times [3.5, 7]$ . We apply the  $\mathcal{C}_{\text{dataset}}$  contractor with  $\mathcal{N}_1$  and  $[\mathbf{x}]$ . Figure 2.47 shows how Algorithm 2 is recursively applied and outputs  $[\mathbf{x}] = [7, 11] \times [3.5, 6]$ .

*Remark 2.121.* The dataset contractor  $\mathcal{C}_{\text{dataset}}$  can be used to evaluate the hull of currents (or outer approximation of all currents) in a given area. Let us assume we have a paving of our area  $\mathcal{Q}$  of  $\mathbb{R}^2$  and an interval staircase function  $\Phi$  of the currents associated to  $\mathcal{Q}$ . We want to compute a hull of the current in an area  $[\mathbf{a}] \in \mathbb{R}^2$ . We build the interval vector of  $\mathbb{R}^{2 \times 2}$ ,  $[\mathbf{x}] = [\mathbf{a}] \times [-\infty, +\infty]^2$  meaning that we do not know the current in the area. We can then apply the contractor  $\mathcal{C}_{\text{dataset}}$  to  $[\mathbf{x}]$  to obtain an over approximation of the current. This contractor can be viewed as a tool which replaces the evaluation of an inclusion function. Indeed, inclusion functions are usually built with an analytical expression which is not the case here as we have discrete datasets.

The contractor can also work the other way: if we do not know our position, but we have a measure of the velocity of currents, we can contract the position.

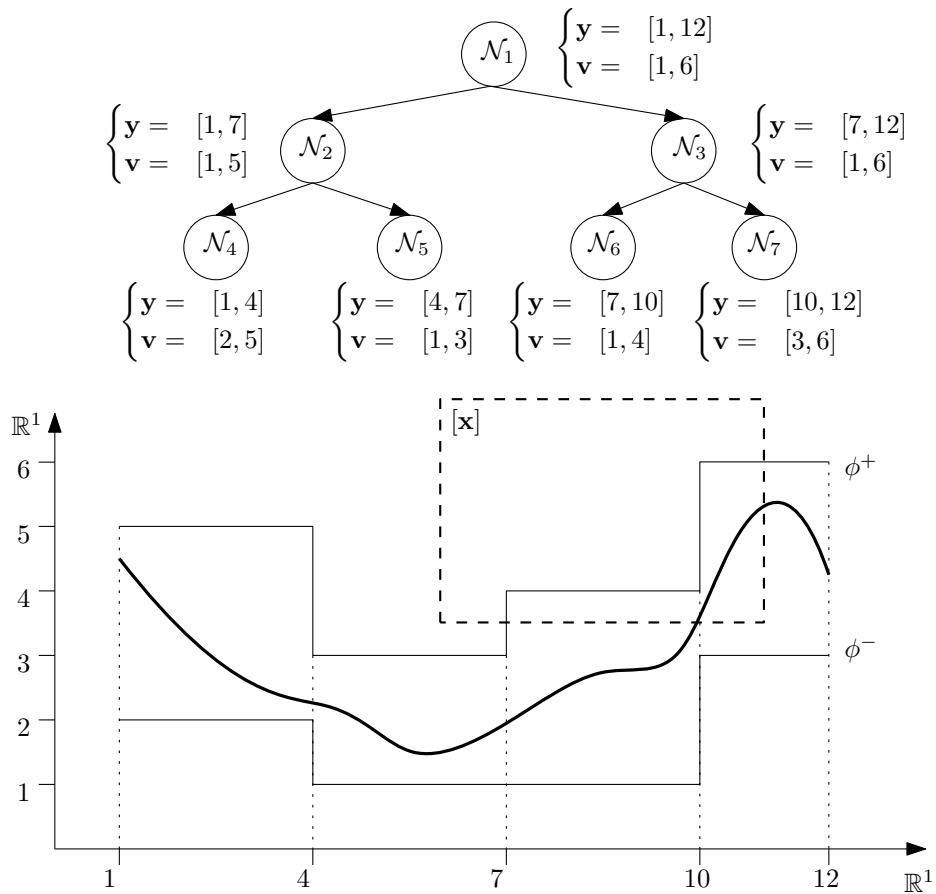


Figure 2.46: Example of a dataset and its tree representation using an interval staircase function on a subpaving.



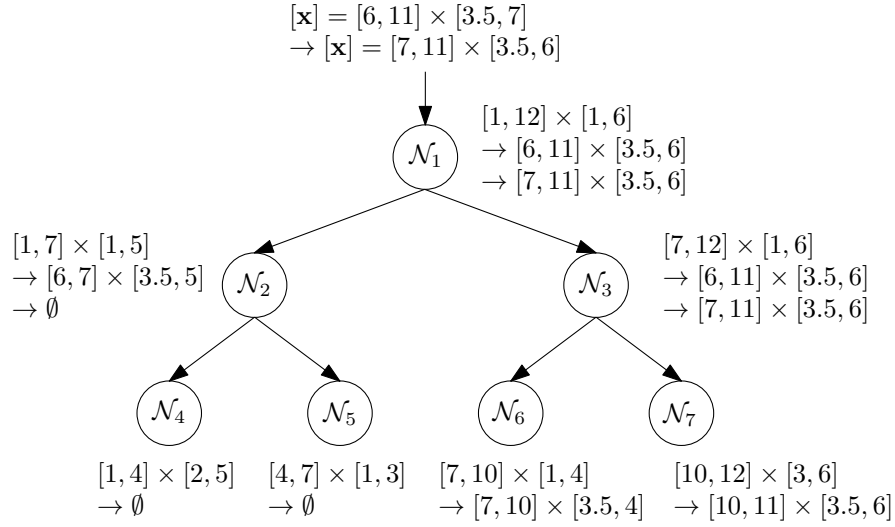


Figure 2.47: An illustration of how the  $\mathcal{C}_{\text{dataset}}$  is applied on the box  $[\mathbf{x}]$  of Figure 2.46 using Algorithm 2.

## 2.4 Conclusion

The aim of this chapter was to provide the tools to guarantee that the path assigned to a robot is safe. We recall that such a path can be more complex than a geographical path  $(x, y)$  and can include any relevant states of the robot (energy, velocity, ...). We have presented the dynamical system theory which will be used to model the behavior of our robot. We have also defined feasible paths in such systems. In addition, we have seen how particular sets of paths, the invariants sets, can handle the behavior of dynamical systems. As we want to analyze global properties of such systems, we have proposed to adopt an Eulerian approach which is consistent with the study of paths instead of trajectories.

Two main questions were then raised: how to handle complex object such as paths, in a computer, and how to validate that a path fulfills a set of constraints. For the first question, we have introduced the framework of Abstract Interpretation (AI) that aims to study how complex objects can be represented in a simpler form, *i.e.* can be abstracted, and therefore handled in a computer. We have presented several classic abstract domains such as intervals, convex polytopes and subpavings of boxes. For the second question, we have introduced the framework of Constraint Programming (CP) that formalizes how a problem can be solved by applying constraints to domains. For both questions, we have highlighted how tools from different communities could be gathered to efficiently address our problem. We have

tried to draw parallels between the communities of Abstract Interpretation, Constraint Programming and Interval Analysis. Finally, we have provided some additional tools such as the dataset contractor that will be helpful to handle discrete data such as the outputs of ocean currents forecasts.

However, we have seen that working with paths of dynamical systems is very complex. Indeed, there exists no dedicated abstract domain to handle paths or invariant sets. Classic abstract domains are not well suited for applying constraints such as “the path should not collide the ground”. Therefore, the next chapter will be dedicated to the presentation of a new domain dedicated to paths. This point is crucial as it will allow to formalize much more efficiently safety problems.

# Chapter 3

## Mazes: a new abstract domain for paths

### Contents

---

<b>3.1</b>	<b>Definitions and problem statement . . . . .</b>	<b>82</b>
<b>3.2</b>	<b>Mazes . . . . .</b>	<b>83</b>
3.2.1	Roads . . . . .	84
3.2.2	Inclusion and lattice . . . . .	85
3.2.3	Door consistency . . . . .	87
<b>3.3</b>	<b>Method and algorithm . . . . .</b>	<b>88</b>
3.3.1	Computing an outer approximation for $\text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ . . . . .	89
3.3.2	Computing an inner approximation for $\text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ . . . . .	91
3.3.3	Main Algorithm . . . . .	97
3.3.4	The Van Der Pol example . . . . .	103
3.3.5	Parameters that affect the speed of convergence . . . . .	103
<b>3.4</b>	<b>Toward an implementation . . . . .</b>	<b>108</b>
3.4.1	The sliding issue . . . . .	108
3.4.2	Using abstract domains without the ACC . . . . .	114
3.4.3	Constraint propagation heuristic and multithreading capabilities . . . . .	114
3.4.4	Using existing libraries . . . . .	116
<b>3.5</b>	<b>Differential inclusion &amp; system with input . . . . .</b>	<b>119</b>
3.5.1	Outer approximation . . . . .	119
3.5.2	Inner approximation . . . . .	120

3.5.3	The example of the reverse Van Der Pol system with an input . . . . .	122
<b>3.6</b>	<b>Conclusion . . . . .</b>	<b>123</b>

---

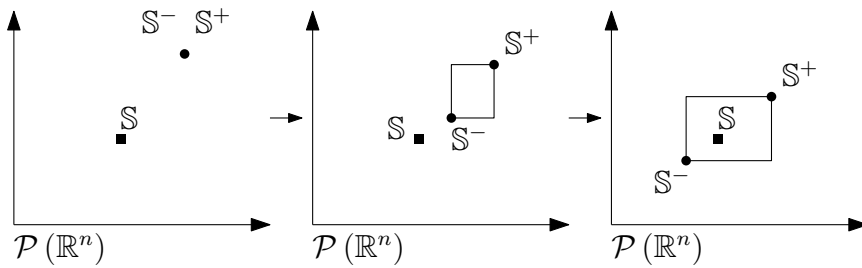


Figure 3.1: Applying constraint techniques to obtain a bracket of a solution set:  $\mathbb{S}^- \subset \mathbb{S} \subset \mathbb{S}^+$ .

To validate the safety of a robot’s path, we need to be able to handle sets of paths in a computer. We have seen from the previous chapter that classic abstract domains were not suited to handle sets of paths. Indeed, applying constraints such as “the path should stay forever inside a set” appears to be complex using classic domains. This is why, this chapter focuses on presenting a new abstract domain for paths that will be called a *maze*.

Along with their formalization, we will show how *mazes* can be used to solve the problem of bracketing *largest positive invariant sets*. We have chosen to focus on this problem as largest positive invariant sets can be the cornerstone of a set of classic problems which deal with the validation of dynamical systems properties. This will allow to apply complex constraints on sets of paths.

More formally, we want to bracket the largest positive invariant set<sup>1</sup>  $\mathbb{S} = \text{Inv}_f^+(\mathbb{X})$  (see Definition 2.26 on page 27) of a set  $\mathbb{X}$  associated to an autonomous dynamical system  $\mathcal{S}$  of the form  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ . This set exists and is unique as positive invariant sets have a lattice structure (see Remark 2.24 on page 26). We recall that solving this problem means to find two sets  $\mathbb{S}^+$  and  $\mathbb{S}^-$  such that  $\mathbb{S}^- \subset \text{Inv}_f^+(\mathbb{X}) \subset \mathbb{S}^+$  (see Definition 2.104 on page 62). The overall idea of this chapter is to apply constraint techniques to two sets  $\mathbb{S}^+$  and  $\mathbb{S}^-$  until a bracket of  $\mathbb{S}$  is obtained, as illustrated in Figure 3.1.

In the literature, corresponding methods to compute invariant sets are generally restricted to linear dynamics (Rakovic et al., 2005; Tahir and Jaimoukha, 2012).

<sup>1</sup>In this chapter, we will shorten for ease of clarity *positive path* in *path* as we will only focus on bracketing largest positive invariant sets.

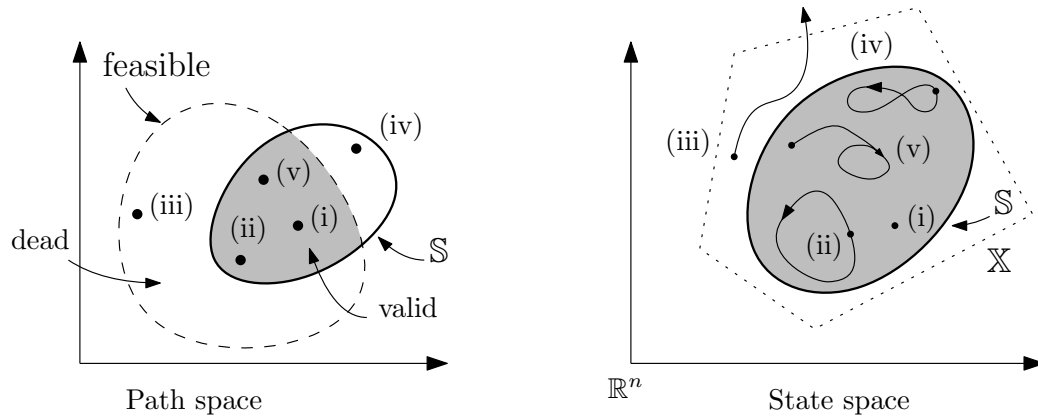


Figure 3.2: Example of paths in the path space and their abstraction in the state space. *Valid* paths correspond to paths that are solution of  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$  and that belong to  $\mathbb{S}$ .

### 3.1 Definitions and problem statement

**Valid and dead paths** We have defined in Section 2.1.2 the notion of *feasible* path associated to a dynamical system. As we now consider the problem of finding paths that belong to the largest positive invariant set of a set  $\mathbb{X}$ , we will give a name to such paths.

**Definition 3.1** (Valid and dead paths). Let us consider a *feasible* path (see Definition 2.10 on page 20). It is *valid* if all its points belong to  $\mathbb{X}$ , otherwise it is *dead*.

**Example 3.2.** We will illustrate here the notion of *valid* paths. We first recall that the set  $\mathbb{S}$  belongs to  $\mathbb{R}^n$ , the state space. To know if a path is valid, we have to use both the evolution function (feasibility) and the set  $\mathbb{X}$  (membership). Figure 2.10 (left) represents the two conditions in the path space. Note that this space is of infinite dimension as there exists an infinite number of paths for every initial condition in  $\mathbb{R}^n$ . In this representation, every point in this space corresponds to a unique path.

We will consider now several examples. The paths (i), (ii) and (v) are *valid* as they are feasible and are included in  $\mathbb{S}$ . The path (iii) is feasible but is *dead* as it does not belong to  $\mathbb{X}$ . The path (iv) belongs to  $\mathbb{X}$  but is not *feasible*.

We can see that the only knowledge of the set  $\mathbb{S}$  is therefore not sufficient to determine if a path is *valid* as we have also to take into account the evolution function.  $\text{Inv}_f^+(\mathbb{X})$  is indeed an abstraction in  $\mathbb{R}^n$  of *valid* paths.

**Constraint network** The main idea we will develop in this chapter, to bracket from inside and outside the set  $\mathbb{S}$ , is to build two complementary constraint networks respectively  $\mathcal{H}$  and  $\overline{\mathcal{H}}$ . An element  $\mathbf{x}$  of the domain of the CN will be therefore either a solution of  $\mathcal{H}$  or a solution of  $\overline{\mathcal{H}}$ . By applying a constraint propagation alternatively on  $\mathcal{H}$  and  $\overline{\mathcal{H}}$ , we will be able to build an inner and an outer approximation of the solution set.

For our problem, to apply a constraint approach, we need to define the constraint network  $\mathcal{H}$ :

- The variables  $\mathcal{V}$  are the paths which are consistent with  $\mathcal{S} : \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ .
- The unique constraint is the following: “the path is valid”, *i.e.* it should belong to  $\mathbb{X}$ .

The complementary CN,  $\overline{\mathcal{H}}$ , has the same variables as  $\mathcal{H}$  except that the constraint is on the contrary: “the path is dead”, *i.e.* it does not strictly belong to  $\mathbb{X}$ .

We have now defined the variables and the constraints, it remains to define the domains for paths which is the goal of the next section.

## 3.2 Mazes

In the literature, no type of domains has been proposed to enclose paths. We propose here a new domain called *maze*. Ideally, a domain should be representable in the memory of a computer which means that it has a finite number of states. It should also have a lattice structure in order to allow intersections and convergence proofs of algorithms (see Theorem 2.98 on page 59). Moreover, choosing a domain that verifies the ACC will avoid the use of a widening operator.

In this section, we will give the definitions of the different objects that form a *maze* and we will look at the basic properties of this new domain.

**Definition 3.3** (Maze). A *maze*  $\mathcal{L}$  of  $\mathbb{R}^n$  is composed of:

- a paving  $\mathcal{P}$ , *i.e.*, a union of non overlapping boxes which covers  $\mathbb{R}^n$ ;
- a set of non-overlapping *doors* between adjacent boxes. A path is allowed to go from one box to another adjacent box by crossing a *door*. Formally, a *door*  $\mathcal{D}$  is defined as an abstract domain in a subspace of dimension  $n - 1$  included in the intersection between two adjacent boxes of dimension  $n$ .





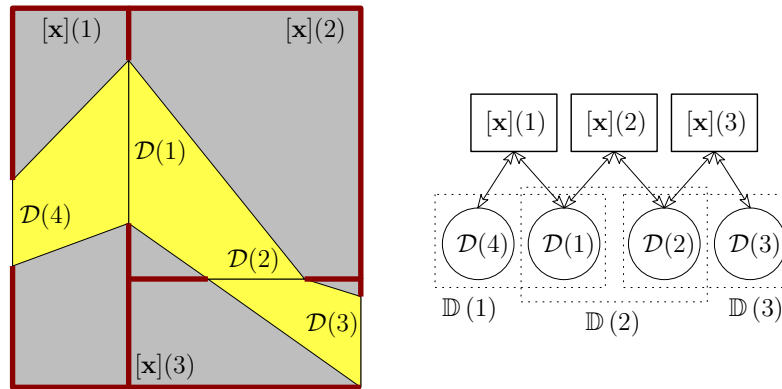


Figure 3.4: Left: A maze made with three roads, Right: the machine representation of the links between boxes and doors of the maze.

*Remark 3.10.* For a given box  $[\mathbf{x}]$ , the set of possible roads  $\langle [\mathbf{x}], \mathbb{D} \rangle$  is a complete lattice with respect to the inclusion:

- For the largest element  $\langle [\mathbf{x}], \top \rangle$ , all points of the boundary of  $[\mathbf{x}]$  belong to a door. We say that all doors are *open* or equivalently that there is no wall.
- For the least element  $\langle [\mathbf{x}], \perp \rangle$ ,  $\mathbb{D}$  is empty and we say that all doors are *closed*.

**Definition 3.11** (Convex hull). Given a road  $\langle [\mathbf{x}], \mathbb{D} \rangle$ , we define  $\text{CONV}(\mathbb{D})$  as the polytope corresponding to the convex hull of  $\mathbb{D}$ .

**Example 3.12.** In Figure 3.3 the yellow polytopes correspond to  $\text{CONV}(\mathbb{D})$  for each road of the maze. Figure 3.4 depicts a maze with three roads:  $\langle [\mathbf{x}](1), \{\mathcal{D}(1), \mathcal{D}(4)\} \rangle$ ,  $\langle [\mathbf{x}](2), \{\mathcal{D}(1), \mathcal{D}(2)\} \rangle$ ,  $\langle [\mathbf{x}](3), \{\mathcal{D}(2), \mathcal{D}(3)\} \rangle$ . We have  $\text{DOORS}([\mathbf{x}](1)) = \mathcal{D}(1) \cup \mathcal{D}(4)$ ,  $\text{DOORS}([\mathbf{x}](2)) = \mathcal{D}(1) \cup \mathcal{D}(2)$  and  $\text{DOORS}([\mathbf{x}](3)) = \mathcal{D}(2) \cup \mathcal{D}(3)$ .

### 3.2.2 Inclusion and lattice

**Definition 3.13** (Inclusion). Given two mazes  $\mathcal{L}_a$  and  $\mathcal{L}_b$ . We say that  $\mathcal{L}_a$  is included in  $\mathcal{L}_b$ , denoted by  $\mathcal{L}_a \subset \mathcal{L}_b$ , if

- the boxes of  $\mathcal{L}_a$  are sub-boxes of the boxes of  $\mathcal{L}_b$  and
- all paths in  $\mathcal{L}_a$  also belong to  $\mathcal{L}_b$ .

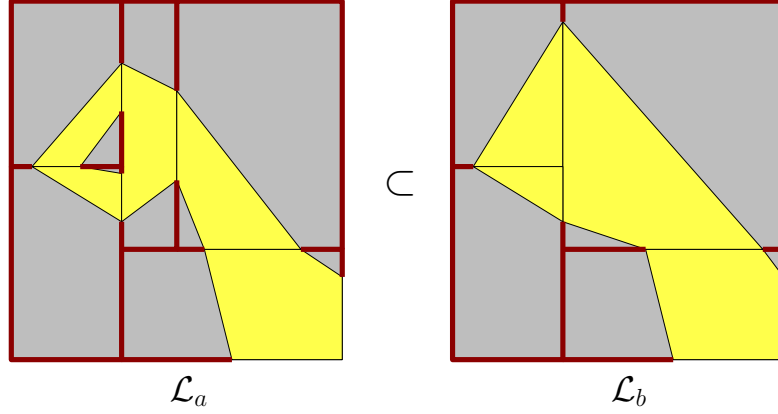


Figure 3.5: Inclusion between two mazes.

**Example 3.14.** An illustration of this definition is given by Figure 3.5 where the left maze contains less paths than the right maze. Indeed, the left maze  $\mathcal{L}_a$  is included in the right maze  $\mathcal{L}_b$ , since (i) the 5 boxes of  $\mathcal{L}_a$  are all sub-boxes of the 3 boxes of  $\mathcal{L}_b$  and (ii) all doors of  $\mathcal{L}_a$  are tighter than those of  $\mathcal{L}_b$ . Moreover, it is trivial to check that if  $\mathcal{L}_a \subset \mathcal{L}_b$  then all paths in  $\mathcal{L}_a$  are also in  $\mathcal{L}_b$ , but there is no equivalence.

**Definition 3.15** (Operations). Given two mazes  $\mathcal{L}_a$  and  $\mathcal{L}_b$ ,

- the *meet*  $\mathcal{L}_a \cap \mathcal{L}_b$  is define as the largest maze (with respect to  $\subset$ ) which is included in both  $\mathcal{L}_a$  and  $\mathcal{L}_b$ ;
- the *join*  $\mathcal{L}_a \cup \mathcal{L}_b$  is define as the smallest maze which contains both  $\mathcal{L}_a$  and  $\mathcal{L}_b$ .

It is trivial to check that the set of mazes of  $\mathbb{R}^n$  is a lattice.

*Remark 3.16.* Given a maze  $\mathcal{L}$  of  $\mathbb{R}^n$  and a set  $\mathbb{X} \subset \mathbb{R}^n$ ,  $\mathcal{L} \subset \mathbb{X}$  if the projection on  $\mathbb{R}^n$  of all paths of  $\mathcal{L}$  are included in  $\mathbb{X}$ . Similarly,  $\mathbb{X} \subset \mathcal{L}$  if all paths included in  $\mathbb{X}$  and consistent with  $\mathcal{S}$  are included in  $\mathcal{L}$ .

*Remark 3.17.* In practice, for two mazes built on the same subpaving, the meet, respectively the join, operator corresponds to the meet, respectively the join, operator of corresponding doors. With a door represented by an abstract domain such as boxes or polytopes, the join operator at the level of each door will be the convex hull operator. In that case, we will use the symbol  $\sqcup$  for the *join* operation where  $\mathcal{L}_a \cup \mathcal{L}_b \subseteq \mathcal{L}_a \sqcup \mathcal{L}_b$ . The same rule can be applied if we only have an over approximation of the *meet* operator for the abstract domain:  $\mathcal{L}_a \cap \mathcal{L}_b \subseteq \mathcal{L}_a \sqcap \mathcal{L}_b$ .

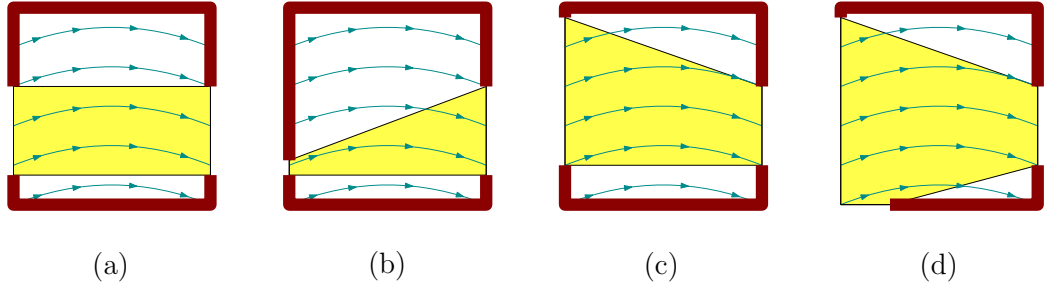


Figure 3.6: The roads (b), (c), (d) are door-consistent. The polygon  $\text{CONV}(\mathbb{D})$  is painted yellow.

### 3.2.3 Door consistency

**Definition 3.18** (Door consistency). The road  $\langle [\mathbf{x}], \mathbb{D} \rangle$  is said to be *door-consistent* with the system  $\mathcal{S}$  if all paths in  $[\mathbf{x}]$ , that are consistent with the doors  $\mathbb{D}$  and with the state equation, remain inside  $\text{CONV}(\mathbb{D})$ .

*Remark 3.19.* This property means that inside  $[\mathbf{x}]$  a trajectory cannot leave  $\text{CONV}(\mathbb{D})$  and then come back. The concept is illustrated by Figure 3.6. The road (a) is not door-consistent since some trajectories which are consistent with the two doors of  $\mathbb{D}$  may leave  $\text{CONV}(\mathbb{D})$ . The left door of road (a) may be contracted into road (b) to make the road door-consistent. It may also be inflated into roads (c) or (d) to get the door-consistency. We can notice that the inflation proposed by road (c) is more accurate than that of road (d).

*Remark 3.20* (Implementation). We do not memorize the yellow polytope associated with the road  $\langle [\mathbf{x}](i), \mathbb{D}(i) \rangle$ , but only the doors, since it can be obtained geometrically by using as a first estimate the outer approximation of the vector field  $[\mathbf{f}]([\mathbf{x}])$  in the box.

The yellow polytope corresponds to the set of all point  $\mathbf{a} \in [\mathbf{x}]$  such that there exists a ray starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [\mathbf{f}]([\mathbf{x}])$  that intersects  $\text{DOORS}([\mathbf{x}])$ . Figure 3.7 illustrates the principle. The visibility contractor presented in Section 2.3.2.3 on page 65 can be used to obtain the yellow polytope.

We will see in next sections that after reaching a fixed point, all roads will be door-consistent. Therefore  $\text{CONV}(\mathbb{D})$  will be an over approximation of the road, which is not the case if the road is not door consistent. For simplicity, we will use  $\text{CONV}(\mathbb{D})$  to represent the roads in our Figures even if it might be an over approximation.

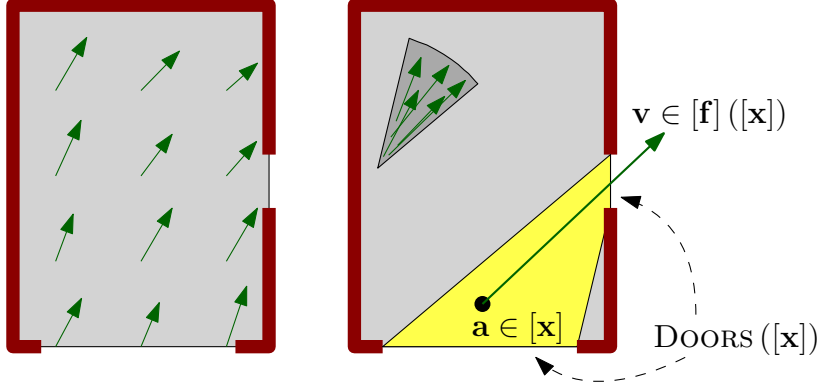


Figure 3.7: Illustration on how the polytope associated to an arbitrary road can be rebuilt from the doors and the state equation.

### 3.3 Method and algorithm

As stated in the introduction, to bracket the largest positive invariant set  $\mathbb{S}$  included in a set  $\mathbb{X}$ , we propose to build two complementary constraint networks. The first one defines the *valid* paths (*i.e.*, the paths that satisfy the state equation and that stay in  $\mathbb{X}$ ) and the second one defines the *dead* paths (*i.e.*, the paths that satisfy the state equation and that leave  $\mathbb{X}$ ).

Recall that, since the system  $\mathcal{S}$  is deterministic, there exists a one-to-one correspondence between each path and the corresponding initial state. This point is important as it justifies that to characterize a *valid* path, we can work with a paving in the space  $\mathbb{R}^n$  of all initial states rather than in the set of paths. Indeed, the latter has an infinite dimension and cannot thus be handled in a computer. In other words, paths of  $\mathcal{S}$  can be soundly abstracted by their initial condition<sup>2</sup>.

We will use mazes as domain of the CN. To bracket  $\mathbb{S} = \text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ , we want to obtain two mazes  $\mathcal{L}_{\mathbb{S}^+}$  and  $\mathcal{L}_{\mathbb{S}^-}$  representing respectively  $\mathbb{S}^+$  and  $\mathbb{S}^-$  such that:

$$\mathcal{L}_{\mathbb{S}^-} \subset \mathbb{S} \subset \mathcal{L}_{\mathbb{S}^+}. \quad (3.1)$$

We will apply contractor or inflator techniques (see Section 2.3.2 on page 62) such that from an initial state of the domain, *i.e.* of the maze, we reach a fixed point that verifies Equation 3.1.

Sections 3.3.1 and 3.3.2 present the method to bracket, respectively the

<sup>2</sup>Lagrangian methods use a different abstraction as they have to additionally bisect in the time space which appears to be not necessary for our problem.

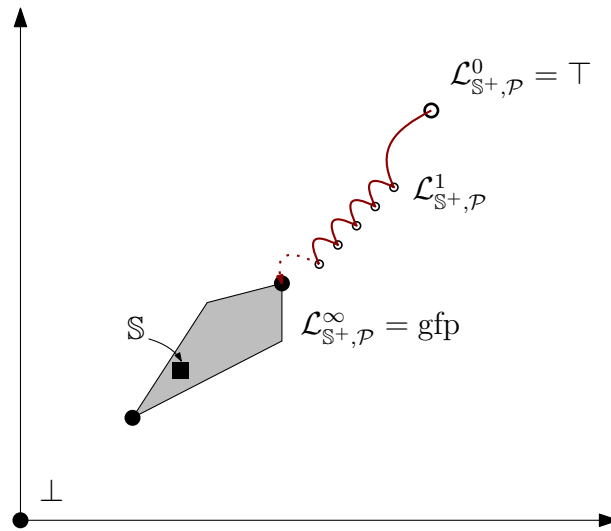


Figure 3.8: Lattice representation of mazes with a given paving  $\mathcal{P}$  and contractions towards a *gfp*. The grey polygon corresponds to the set of fixed points.

outer and the inner approximation, of the set  $\mathbb{S}$  with a given paving<sup>3</sup>  $\mathcal{P}$ . Section 3.3.3 combines the two subsections and adds the notion of paving bisection.

### 3.3.1 Computing an outer approximation for $\text{Inv}_f^+(\mathbb{X})$

To compute an outer approximation  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$  for a given paving  $\mathcal{P}$ , we will use a contractor approach by *removing dead paths without removing any valid paths* (first *strategy* of Section 2.3.2 on page 62).  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$  will be initialized to top, *i.e.* all roads equal to  $\langle [\mathbf{x}], \top \rangle$ .

Figure 3.8 illustrates the contractions of the maze starting from an initial maze  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^0$  that reaches a *gfp*.

We need now to define the contractors that will remove *dead* paths from the domain. We introduce therefore the two following contractors: the *boundary condition contractor* and the *flow contractor*.

**Boundary condition contractor** ( $\text{BOUNDARYCONTRACT}_{\mathbb{X}}(\mathcal{L})$ ) For each road of the maze, if the corresponding box  $[\mathbf{x}]$  has an empty intersection with  $\mathbb{X}$  then we contract the road to  $\langle [\mathbf{x}], \perp \rangle$ , *i.e.* we close all doors linked to  $[\mathbf{x}]$ .

<sup>3</sup>In practice, we will assume that the paving  $\mathcal{P}$  is built such that there exists at least one box  $[\mathbf{x}]$  of the paving that is included in  $\mathbb{X}$  in order to avoid a particular case.

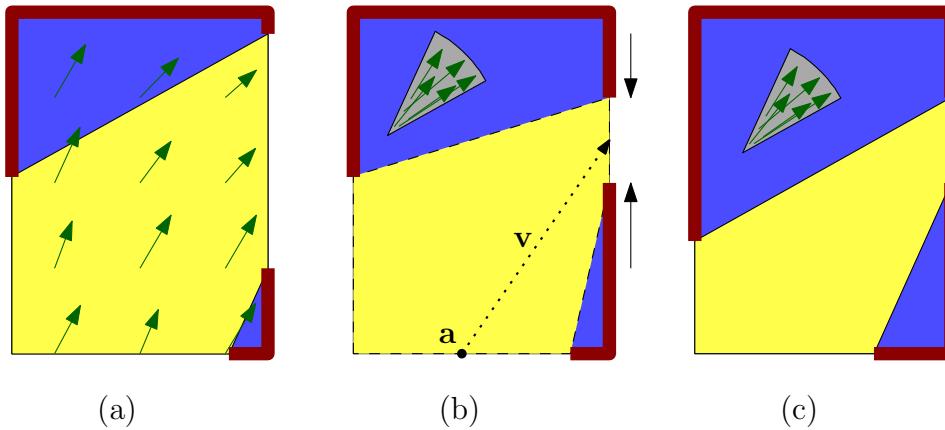


Figure 3.9: Illustration of the flow contractor; the road is contracted backward and the eliminated zones are in blue.

**Flow contractor** ( $\text{FLOWCONTRACTBWD}_f(\mathcal{L})$ ) We contract each road  $\langle [\mathbf{x}], \mathbb{D} \rangle$  of the maze with respect to the constraint  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ . To do this, we contract the doors without losing any point in the set of *all*  $\mathbf{a} \in \text{DOORS}([\mathbf{x}])$  such that there exists a ray starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [\mathbf{f}]([\mathbf{x}])$  that does not intersect  $\text{WALLS}([\mathbf{x}])$ . The contractor<sup>4</sup> should be applied only if the doors are not linked with a box  $[\mathbf{x}]$  where  $\mathbf{0} \in [\mathbf{f}]([\mathbf{x}])$ .

The flow contractor operates a backward contraction as a door is contracted in function of the future of the possible trajectories. It also removes only dead paths and none of the valid paths as the vector flow is over approximated by  $[\mathbf{f}]([\mathbf{x}])$ .

**Example 3.21.** An illustration of the corresponding contractor is given by Figure 3.9. (a) illustrates a road with the vector field  $\mathbf{f}(\mathbf{x})$ , three doors, and  $\text{CONV}(\mathbb{D})$  (the yellow polytope). In (b) an external event comes to contract the right door (black arrows facing each other). The vector field is now represented by the gray cone  $[\mathbf{f}]([\mathbf{x}])$  computed using interval arithmetic (see Section 2.2.2.1). The contraction of the bottom and left doors, and the updated polytope  $\text{CONV}(\mathbb{D})$  are represented in (c). We can check that after the contraction, the road is door-consistent since no trajectory can enter in  $\text{CONV}(\mathbb{D})$ .

*Remark 3.22.* The constraint is purely geometrical and do not use any time integration. It can be seen as a zero order approximation of the vector flow. We here focus on the path and not on the trajectory. Indeed, we do not need

<sup>4</sup>Details about how the implementation of such contractor can be performed is given later in Section 3.4.4.

to know at what time the trajectory will cross or will never cross a door but only if there exists an intersection for any time in the future.

**Propagation** We then combine both contractors by applying the boundary condition contractor and the flow contractor several times to each road until a fixed point is reached.

**Example 3.23.** Figure 3.10 illustrates an example of the propagation. In Sub-figure (1), all doors of all roads are assumed to be open. Since the blue box in Sub-figure (2) is outside  $\mathbb{X}$  (dashed line) all corresponding doors are closed. In Sub-figures (3)-(6) the propagation contracts backward the roads without eliminating any valid path, which are all trapped inside the yellow zone representing the current maze.

Algorithm 3 summarizes the steps to compute the outer approximation of the largest invariant sets of the set  $\mathbb{X}$ .

---

<b>Algorithm 3:</b>	Computing the outer approximation (COMPUTEOUTER).
---------------------	---

---

<b>Input:</b>	$\mathcal{P}, \mathbf{f}, \mathbb{X}$
<b>Output:</b>	$\mathbb{S}^+$
1	$\mathbb{S}^+ \leftarrow \emptyset;$
2	$\mathcal{L}_{\mathbb{S}^+, \mathcal{P}} \leftarrow \{ \langle [\mathbf{x}], \top \rangle \mid [\mathbf{x}] \in \mathcal{P} \};$
3	$\text{BOUNDARYCONTRACT}_{\mathbb{X}}(\mathcal{L}_{\mathbb{S}^+, \mathcal{P}});$
4	<b>repeat</b>
5	$\text{FLOWCONTRACTBWD}_{\mathbf{f}}(\mathcal{L}_{\mathbb{S}^+, \mathcal{P}});$
6	<b>until</b> <i>fixed point reached</i> ;
7	<b>foreach</b> $\langle [\mathbf{x}], \mathbb{D} \rangle \in \mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$ <b>do</b>
8	$\mathbb{S}^+ \leftarrow \mathbb{S}^+ \cup \text{CONV}(\mathbb{D});$
9	<b>end</b>

---

*Remark 3.24.* Both contractors are monotonic functions and the domain is a complete lattice. If we chose a domain for the doors that fulfills the ACC condition then the *gfp*, i.e.  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^{\infty}$ , can be reached after a finite number of steps. This is the case for instance with boxes on floating-point numbers  $\mathbb{IF}^n$ .

### 3.3.2 Computing an inner approximation for $\text{Inv}_{\mathbf{f}}^+(\mathbb{X})$

To compute the inner approximation, we want to obtain a maze  $\mathcal{L}_{\mathbb{S}^-, \mathcal{P}}$  such that it contains no *dead* paths and only *valid* paths. Similarly to the outer

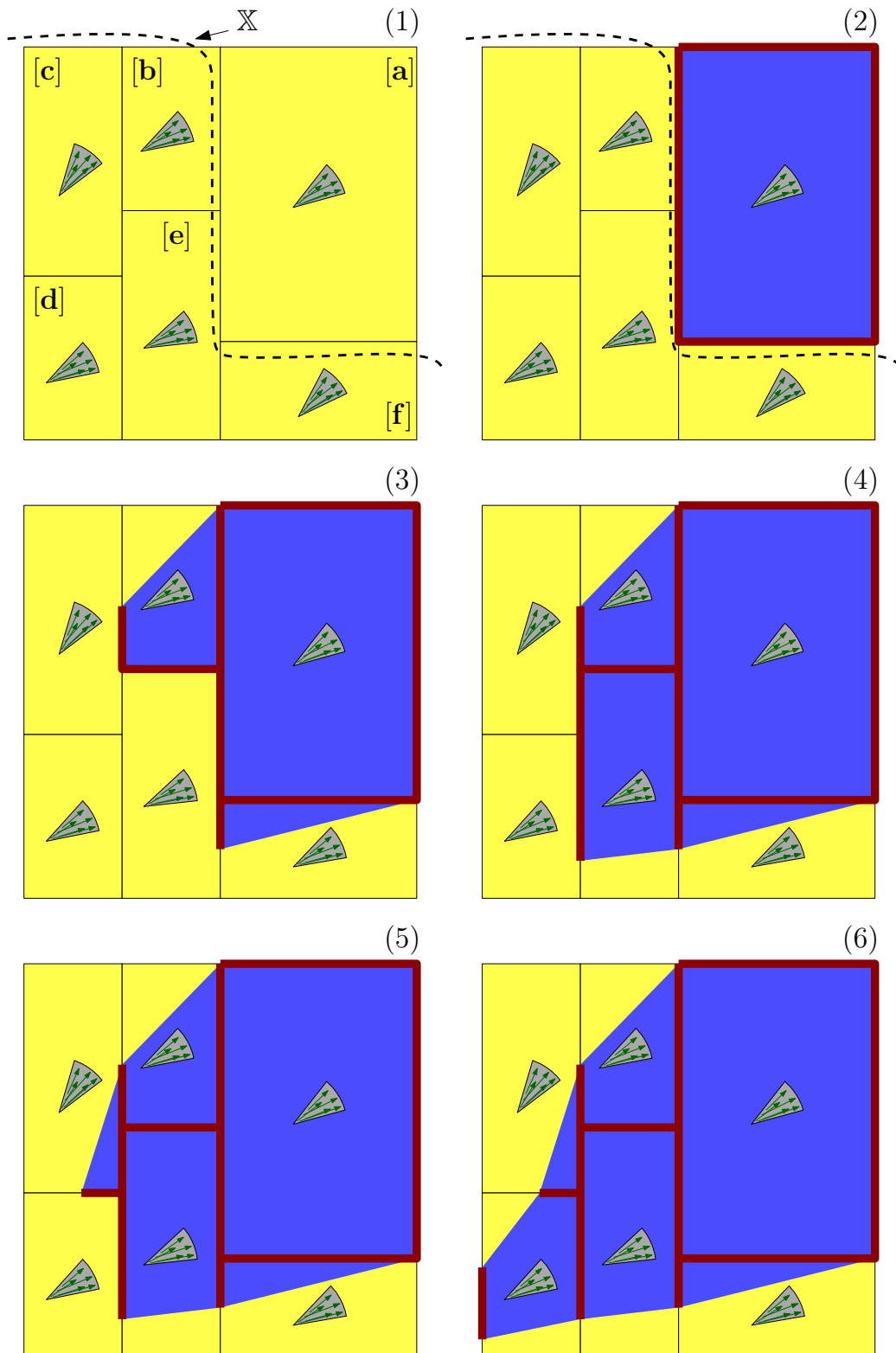


Figure 3.10: Outer contraction from step (1) to (6). At each step, the yellow area is contracted and is necessary a superset of  $\mathbb{S}$ , even if we have not reached the fixed point yet.



approximation, we could have used contraction techniques by *removing all dead paths* (second approach of Section 2.3.2 on page 62). The flow contractor condition would have been changed to “*the set of all  $\mathbf{a} \in \text{DOORS}([\mathbf{x}])$  such that all rays starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [\mathbf{f}]([\mathbf{x}])$  does not intersect  $\text{WALLS}([\mathbf{x}])$ ”.*

However, this contractor is difficult to implement because of the constraint “*all rays*”. We have chosen instead to use the complementary constraint network. This amounts to compute an over approximation  $\overline{\mathbb{S}}^+$  of the complementary set  $\overline{\mathbb{S}}$  which is equivalent to compute an inner approximation  $\mathbb{S}^-$  of  $\mathbb{S}$  where:

$$\overline{\mathbb{S}} = \overline{\text{Inv}_{\mathbf{f}}^+(\mathbb{X})} = \{\mathbf{x}(0) \in \mathbb{R}^n \mid \exists t \geq 0, \mathbf{x}(t) \notin \mathbb{X}\}.$$

To compute  $\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}$  for a given paving  $\mathcal{P}$ , we will consider *dead paths* as the solution instead of *valid paths*. We will use an inflator approach which will *add all dead paths* (third strategy of Section 2.3.2 on page 62) to the domain.  $\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}$  will then be initialized to bottom, *i.e.*, all roads equal to  $\langle [\mathbf{x}], \perp \rangle$ .

Figure 3.11 illustrates the inflation of the maze starting from an initial maze  $\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}^0$  and which reaches a *lfp*. The complementary of the *lfp* is then an inner approximation of  $\mathbb{S}$ , *i.e.*  $\overline{\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}^\infty} \subset \mathbb{S}$ , as all dead paths will be enclosed by  $\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}^\infty$ . In this case, the fixed point must absolutely be reached to prove that we have an inner approximation contrary to the outer approximation case. In the contrary, we do not have any proof that  $\mathbb{S} \subset \mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}^\infty$ .

Similarly to the outer approximation, we will define inflators that will add *dead paths* to the domain. We will introduce therefore two inflators: the *boundary condition inflator* and the *flow inflator*.

**Boundary condition inflator ( $\text{BOUNDARYINFLATE}_{\overline{\mathbb{X}}}(\mathcal{L})$ )** For each road of the maze, if the corresponding box  $[\mathbf{x}]$  intersects  $\overline{\mathbb{X}}$  then we inflate the road to  $\langle [\mathbf{x}], \top \rangle$ , *i.e.* we open all doors linked to  $[\mathbf{x}]$ .

**Flow inflator ( $\text{FLOWINFLATEBWD}_{\mathbf{f}}(\mathcal{L})$ )** We inflate each road  $\langle [\mathbf{x}], \mathbb{D} \rangle$  of the maze with respect to the constraint  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ . To do this, we inflate the doors to enclose all points of the set of *all  $\mathbf{a} \in \text{WALLS}([\mathbf{x}])$  such that there exists a ray starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [\mathbf{f}]([\mathbf{x}])$  that intersects  $\text{DOORS}([\mathbf{x}])$* .

As for the flow contractor, we obtain a backward inflation.

**Example 3.25.** An illustration of the corresponding contractor is given by Figure 3.12. (a) illustrates a road with the vector field  $\mathbf{f}(\mathbf{x})$ , two doors, and

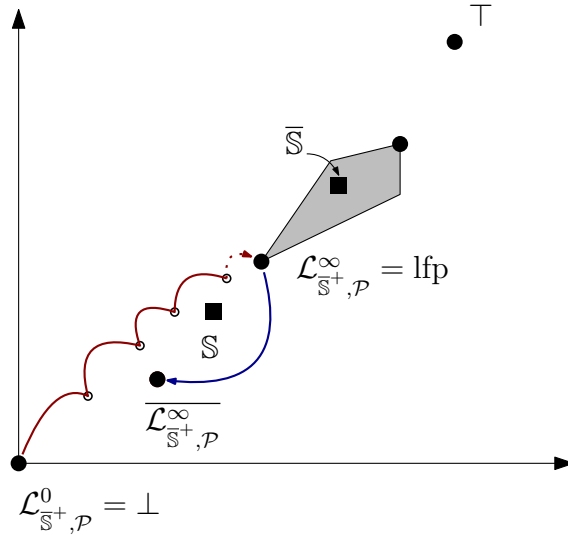


Figure 3.11: Lattice representation of mazes with a given paving  $\mathcal{P}$  and inflation towards a *lfp*. The grey polygon corresponds to the set of fixed points.

$\text{CONV}(\mathbb{D})$  (the yellow polytope). In (b) an external event comes to inflate the door on the right (opposing black arrows). The inflation of the bottom and left doors, and the updated polytope  $\text{CONV}(\mathbb{D})$  are represented in (c). We can check that after the contraction, the road is door-consistent since no trajectory can enter in  $\text{CONV}(\mathbb{D})$ .

**Propagation** Similarly to the outer approximation computation, we combine both inflators by applying the *boundary condition inflator* and the *flow inflator* several times to each road.

**Example 3.26.** Figure 3.13 illustrates the principle up to a fixed point. In Sub-figure (1), all doors of all roads are assumed to be closed. Since the yellow box in Sub-figure (2) intersects  $\mathbb{X}$  (dashed line) all corresponding doors are open. In Sub-figures (3)-(5) the propagation inflates backward the roads by adding all dead paths. Gray areas have not been processed yet. Once the fixed point is reached, we can conclude that the remaining area, is outside  $\bar{\mathbb{S}}$  (painted in magenta), or equivalently that the magenta area corresponds to an inner approximation  $\mathbb{S}^-$  of  $\mathbb{S}$ .

Algorithm 4 summarizes the steps to compute the inner approximation of the largest invariant sets of the set  $\mathbb{X}$ .

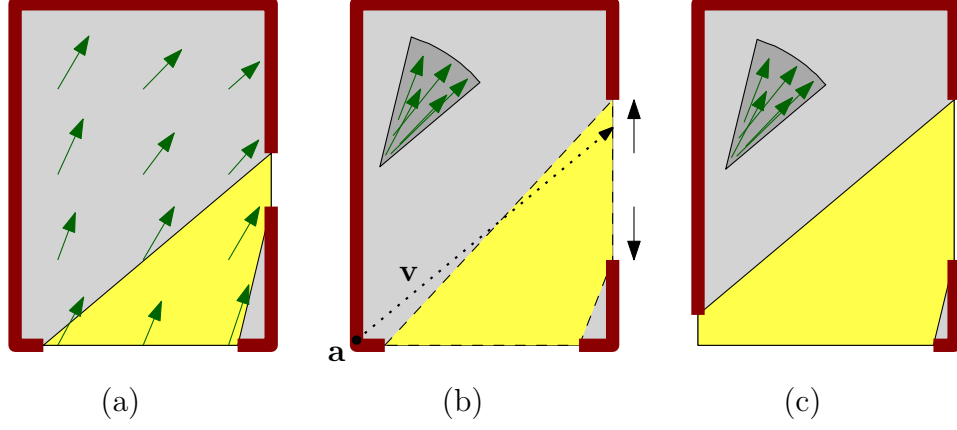


Figure 3.12: Illustration of the flow inflation; the road is inflated backward.

---

**Algorithm 4:** Computing the inner approximation (COMPUTEINNER).

---

**Input:**  $\mathcal{P}, \mathbf{f}, \mathbb{X}$

**Output:**  $\mathbb{S}^-$

- 1  $\mathbb{S}^- \leftarrow \emptyset;$
  - 2  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}} \leftarrow \{ \langle [\mathbf{x}], \perp \rangle \mid [\mathbf{x}] \in \mathcal{P} \};$
  - 3  $\text{BOUNDARYINFLATE}_{\mathbb{X}} \left( \mathcal{L}_{\mathbb{S}^+, \mathcal{P}} \right);$
  - 4 **repeat**
  - 5  $\text{FLOWINFLATEBWD}_{\mathbf{f}} \left( \mathcal{L}_{\mathbb{S}^+, \mathcal{P}} \right);$
  - 6 **until** *fixed point reached*;
  - 7 **foreach**  $\langle [\mathbf{x}], \mathbb{D} \rangle \in \mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$  **do**
  - 8  $\mathbb{S}^- \leftarrow \mathbb{S}^- \cup ([\mathbf{x}] \setminus \text{CONV}(\mathbb{D}));$
  - 9 **end**
-

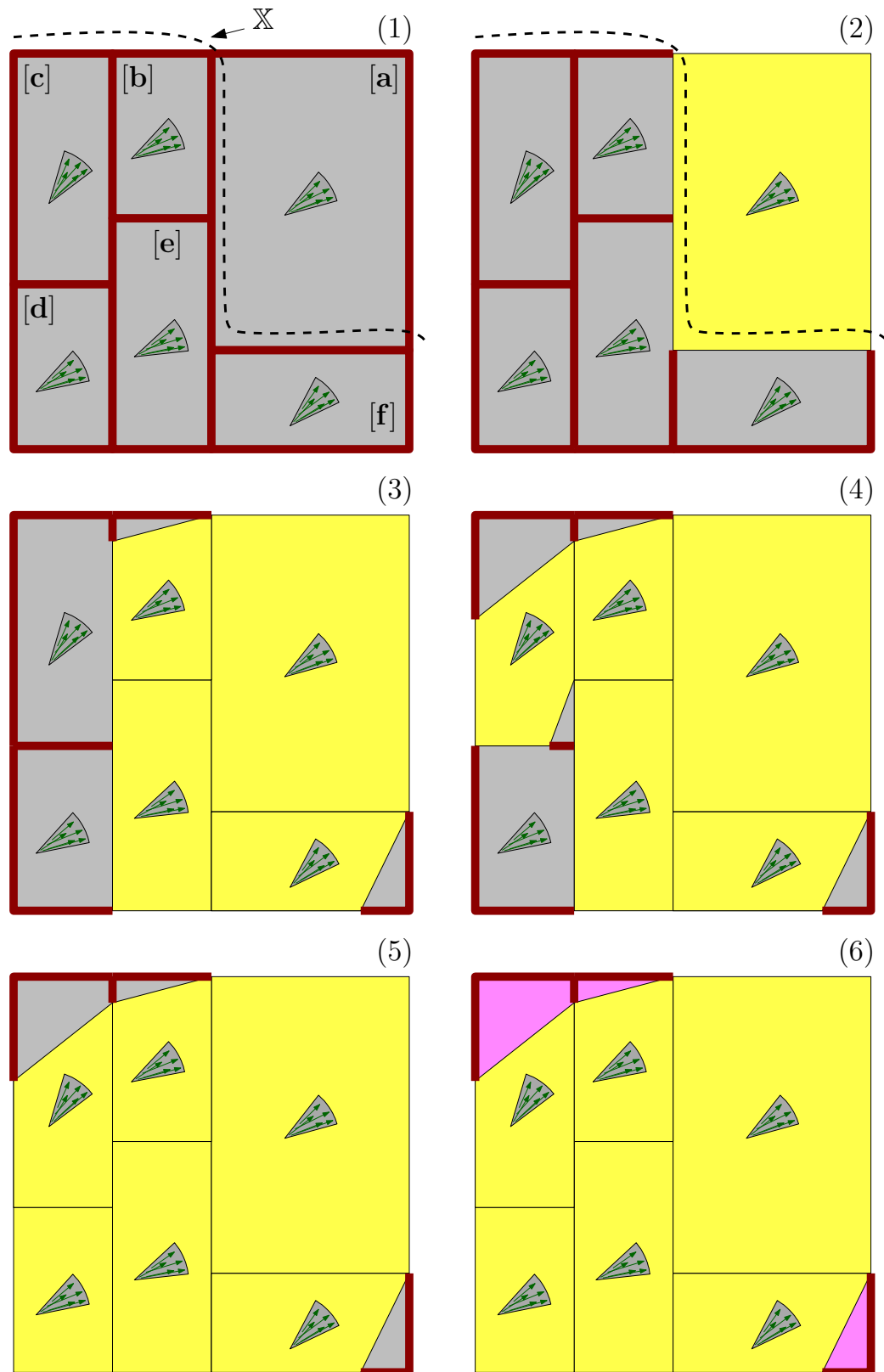


Figure 3.13: Inner inflation from step (1) to step (6). Once the fixed point is reached at step (6), we conclude that the magenta zone is inside  $\mathbb{S}$ .

### 3.3.3 Main Algorithm

We can now combine the computation of the inner and outer approximation to bracket the set  $\mathbb{S}$ . To reduce the number of operations required, we will also add a strategy to bisect gradually the paving  $\mathcal{P}$ . Similarly to the SIVIA algorithm, we will refine the maze only where it is necessary.

#### 3.3.3.1 Paving bisection

The goal of paving bisection is to reduce the over approximation of the vector field which is evaluated for each box  $[\mathbf{x}]$ .

Let us take  $\mathcal{L}_{\mathcal{P}}$ , a maze associated to a given paving  $\mathcal{P}$ , and  $\beta$ , a bisector (see Definition 2.89 on page 55), we can define a sequence of paving  $\mathcal{P}(k)$  such that  $\mathcal{P}(k) = \text{BISECT}(\mathcal{P}(k-1))$  where  $\text{BISECT}$  is an operator that bisects chosen boxes of  $\mathcal{P}$  using  $\beta$ .

In our case, we will chose to bisect only boxes  $[\mathbf{x}]$  that verify  $[\mathbf{x}] \not\subset \overline{\mathbb{S}^-} \cup \mathbb{S}^+$ . In other words, we will not bisect boxes that have been proven to be completely outside  $\mathbb{S}$  (blue area) or completely inside  $\mathbb{S}$  (magenta area). Algorithm 5 bisects the paving according to this constraint.

---

**Algorithm 5:** Bisection of the paving associated to a maze ( $\text{BISECT}$ ).

---

**Input:**  $\mathcal{P}(k-1), \mathbb{S}^-, \mathbb{S}^+$   
**Output:**  $\mathcal{P}(k)$

```

1  $\mathcal{P}(k) \leftarrow \{\emptyset\};$ 
2 foreach  $[\mathbf{x}] \in \mathcal{P}(k-1)$  do
3   if  $[\mathbf{x}] \not\subset \overline{\mathbb{S}^-} \cup \mathbb{S}^+$  then
4      $\mathcal{P}(k) \leftarrow \mathcal{P}(k) \cup \beta([\mathbf{x}]);$ 
5   else
6      $\mathcal{P}(k) \leftarrow \mathcal{P}(k) \cup \{[\mathbf{x}]\};$ 
7   end
8 end
```

---

*Remark 3.27.* Bisectioning the paving will intuitively improve the approximation of the set  $\mathbb{S}$ . Indeed, if we have a convergent inclusion function  $[\mathbf{f}]$  for the system (see Definition 2.76 on page 48), smaller boxes will produce a more accurate evaluation of the vector field inside the box. Given an initial box  $[\mathbf{x}]$  and using  $\beta$  to obtain two bisected boxes  $[\mathbf{x}_1], [\mathbf{x}_2]$ , we have  $([\mathbf{f}]([\mathbf{x}_1]) \cup [\mathbf{f}]([\mathbf{x}_2])) \subseteq [\mathbf{f}]([\mathbf{x}])$ . However, the use of  $\text{BISECT}$  will multiply at most by two the number of boxes to consider, at each iteration. This will result in an exponential growth.

A more bisected paving would reduce under some conditions the *loss of information* largely due to a better evaluation of the system inclusion function. However, we shall verify that we can correctly set the new created doors such that the new bisected maze represents the same set of paths.

**Proposition 3.28.** *Let us take  $\langle [\mathbf{x}], \mathbb{D} \rangle$  a consistent road, and let us consider  $[\mathbf{x}_1], [\mathbf{x}_2]$  the two boxes resulting from the bisection of  $[\mathbf{x}]$ . If  $\text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_1]$  and  $\text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_2]$  are representable in the abstract domain of  $\mathbb{D}$ , then we can set  $\mathbb{D}_1$  and  $\mathbb{D}_2$  as respectively the doors of  $\langle [\mathbf{x}_1], \mathbb{D}_1 \rangle$  and  $\langle [\mathbf{x}_2], \mathbb{D}_2 \rangle$  such that we do not add nor remove paths, i.e.  $\text{CONV}(\mathbb{D}_1) \cup \text{CONV}(\mathbb{D}_2) = \text{CONV}(\mathbb{D})$ .*

*Proof.* The proof is straightforward. We set  $\mathbb{D}_1 = \text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_1]$  and  $\mathbb{D}_2 = \text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_2]$  which is possible as they are both representable in the abstract domain of  $\mathbb{D}$ . We then have  $\text{CONV}(\mathbb{D}_1) \cup \text{CONV}(\mathbb{D}_2) = \text{CONV}(\mathbb{D})$ .  $\square$

**Example 3.29.** To illustrate Proposition 3.28, let us consider several examples. If intervals are used in 2D, then the intersection of  $\text{CONV}(\mathbb{D})$  with a door, which is a degenerated box, is also a degenerated box. In 3D we have to consider polytopes because the use of boxes does not anymore work. To illustrate the issue, let us consider the example of Figure 3.14 where we use boxes as the abstract domain of  $\mathbb{D}$ .  $\text{CONV}(\mathbb{D})$  (in grey) is a polytope and the intersection (in red) with a plane of bisection (in blue) is not anymore a box. In that case, we cannot have  $\text{CONV}(\mathbb{D}_1) \cup \text{CONV}(\mathbb{D}_2) = \text{CONV}(\mathbb{D})$ . Indeed  $\text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_1]$  and  $\text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_2]$  are not representable in the domain of the door. The use of the convex hull operator would be required:  $[\text{CONV}(\mathbb{D}) \cap \partial[\mathbf{x}_1]]$ .

*Remark 3.30.* Another point concerns the bisection heuristic which is critical in our problem. The more the paving will be bisected, the greater the number of operations will be, as more boxes will have to be considered. Bisecting the *right* boxes is therefore important. Characterizing a set, such as  $\mathbb{S}$ , amounts to characterize its boundary. This is why we have chosen to only bisect doors that are on the boundary of the set  $\mathbb{S}$  to save computation time (see the condition  $[\mathbf{x}] \not\subset \overline{\mathbb{S}^-} \cup \mathbb{S}^+$ ).

We can see that the bisection increases the number of different distinct sets of paths that can be represented. This can be seen as a refinement of the abstraction. To illustrate this point, Figure 3.15 shows the process of bisection in the state space (top line) and in the powerset space of feasible paths (bottom line). On the left, we have the initial maze; in the middle, the box is bisected and doors are set accordingly to Proposition 3.28; finally,

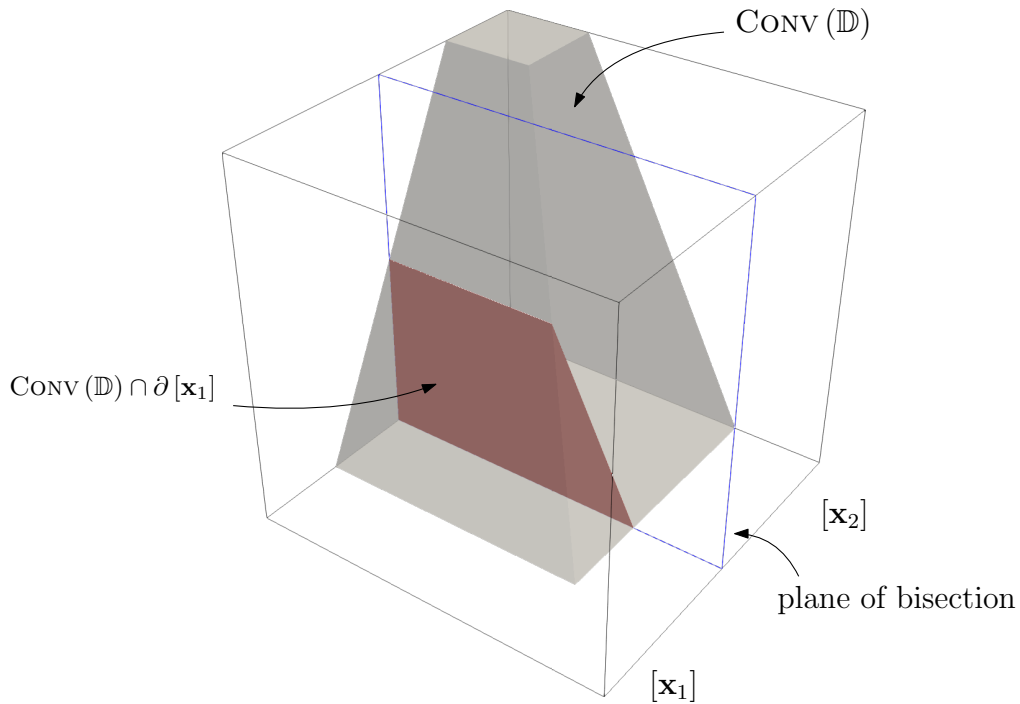


Figure 3.14: Example of bisection in 3D with a closed intersection issue.

on the right, the two roads are contracted. We can see on the powerset space of feasible paths that we have more distinct sets of paths that can be represented by the domain. In fact, the bisection added what could be seen as a new degree of freedom. Therefore, adding doors allows to represent a more accurate approximation of the set  $\mathbb{S}$  or at least to obtain the same approximation.

### 3.3.3.2 An algorithm to bracket the largest positive invariant set

We can now combine the paving bisection and the computation of the outer and inner approximation for a given paving. We propose Algorithm 6 to bracket the largest positive invariant set included in  $\mathbb{X}$ .

The algorithm is initialized with a paving  $\mathcal{P}$  and computes the outer and the inner approximation associated to this paving. If the result is accurate enough it returns the sets  $\mathbb{S}^+$  and  $\mathbb{S}^-$ , otherwise it bisects the paving and recomputes the outer and the inner approximation. The number of time the algorithm loops will be called the number  $k$  of *iterations*.

*Remark 3.31.* Note that to simplify the presented algorithm, the mazes are reset at each loop. In fact, the states of the doors computed at step  $k - 1$

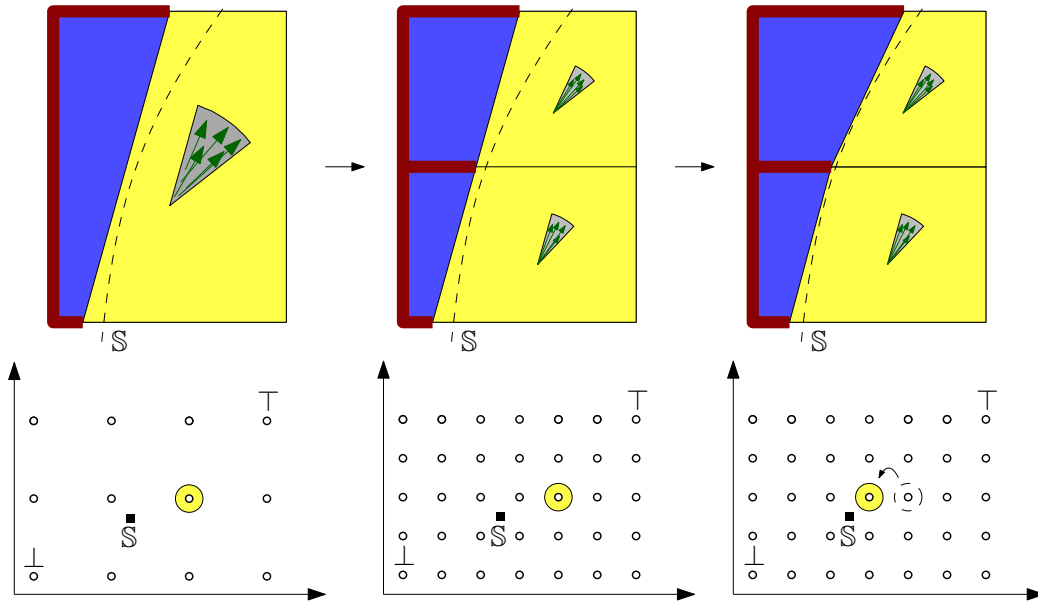


Figure 3.15: Bisection of the paving  $\mathcal{P}$  associated to a maze. Top: the maze, bottom: the associated powerset space of feasible paths.

---

**Algorithm 6:** Bracketing the largest positive invariant set.

---

**Input:**  $f, \mathbb{X}$   
**Output:**  $\mathbb{S}^-, \mathbb{S}^+$

- 1  $\mathcal{P}(0) \leftarrow \{\mathbb{R}^n\};$
- 2  $k \leftarrow 0;$
- 3 **loop**
- 4      $\mathbb{S}^+ \leftarrow \text{COMPUTEOUTER}(\mathcal{P}(k), f, \mathbb{X});$
- 5      $\mathbb{S}^- \leftarrow \text{COMPUTEINNER}(\mathcal{P}(k), f, \mathbb{X});$
- 6     **if** *not accurate enough* **then**
- 7          $\mathcal{P}(k) \leftarrow \text{BISECT}(\mathcal{P}(k-1), \mathbb{S}^-, \mathbb{S}^+);$
- 8          $k \leftarrow k + 1;$
- 9     **else**
- 10         **return**  $\mathbb{S}^-, \mathbb{S}^+;$
- 11     **end**
- 12 **end**

---



could be taken into account for the step  $k$  (see lines 2 and 2 of respectively Algorithms 3 and 4).

Most of the time, the end condition is driven by a number of *iterations* to achieve, instead of an accuracy condition. This guarantees that the algorithm will stop (see later Remark 3.34 on convergence issue).

### 3.3.3.3 Computed enclosure

**Proposition 3.32.** *We have  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty \subseteq \mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty$  and  $\overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty} \supseteq \overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty}$ .*

*Proof.* This proposition is only true in the case where the abstract domain verify the hypothesis of Proposition 3.28. Indeed, given a maze  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty$ , we can build a maze  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^0$  such that for each room we verify Proposition 3.28. As contractions remove non solutions, we will have  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty \subseteq \mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty$ . This is the same for  $\overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty} \supseteq \overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty}$ .  $\square$

**Example 3.33.** To illustrate the algorithm, we show on Figure 3.16, for the outer approximation, the evolution of the maze  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$  subjected to contractions. The *gfp* for each paving are represented by black disks  $\bullet$  while the contractions are represented by circles  $\circ$ . The blue lines correspond to one iteration and the red lines to the contractions presented in Section 3.3.1. At each new iteration, the new *gfp* is then a subset of the previous *gfp*, i.e.  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty \subseteq \mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty$ .

*Remark 3.34.* Proving that the two sequences  $\left(\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty\right)_k$  and  $\left(\overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty}\right)_k$  converge toward  $\mathbb{S}$  has not been done in this work which is not trivial nor true in all cases. We only have that  $\overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty} \subset \mathbb{S} \subset \mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty$  and also that  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty \subset \mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty$  and  $\overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k)}^\infty} \subset \overline{\mathcal{L}_{\mathbb{S}^+, \mathcal{P}(k-1)}^\infty}$ . An uncertainty layer could in fact subsist in the case where we do not have a convergent inclusion function (see Definition 2.76 on page 48). This can be also the case if we do not have an exact fixed point transfer (see Section 2.3.1.2).

### 3.3.3.4 Complexity

Due to the paving bisection, for a given required accuracy, the algorithm has an exponential complexity with respect to the dimension of the state space. This is one of the strongest drawback of this algorithm.

However, the algorithm brackets any non specific and not necessarily convex  $nD^5$  set using facets which is a problem exponential *w.r.t.* the dimension.

---

<sup>5</sup>n-dimensional

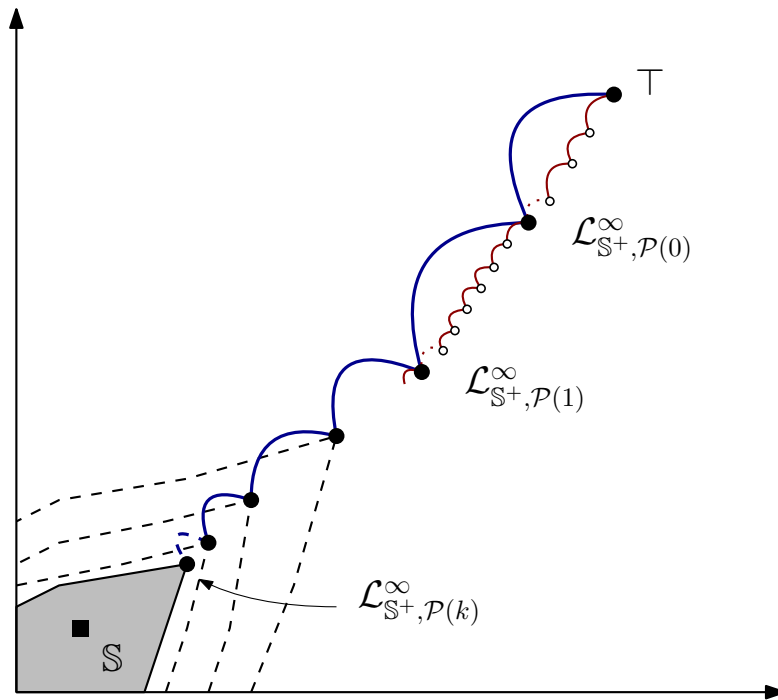


Figure 3.16: Paving refinement and associated *gfp* when computing an outer approximation under hypothesis of Proposition 3.28.

### 3.3.4 The Van Der Pol example

**Largest positive invariant set** In this section, we will apply the algorithm on the Van Der Pol system. We recall the evolution function of this non-linear system:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_2 \\ (1 - x_1^2)x_2 - x_1 \end{pmatrix}.$$

We want to bracket the set  $\mathbb{S} = \text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ . Let us take  $\mathbb{X} = [-3, 3] \times [-3, 3]$ . Figure 3.17 shows<sup>6</sup> the two sets  $\mathbb{S}^-$  (in magenta) and  $\mathbb{S}^+$  (magenta and yellow) at different steps  $k$ . We can note that we need to reach step 11 to obtain a non-empty inner approximation of the set  $\mathbb{S}$ . We can verify that there is always a yellow boundary between the states that have been proved to be outside  $\mathbb{S}$  (in blue) and states that have been proven to be inside  $\mathbb{S}$  (magenta). This boundary is called the *uncertainty layer*.

**Computation time** For each iteration step, we can measure the computation time and the volume of the uncertainty layer. Figure 3.18 shows the result with a logarithm scale. As expected, the computation time is exponential with the iteration because of bisections. We clearly see the effect of the non empty inner approximation after step 11 that considerably reduces the volume of the uncertainty layer. This also allows to reduce the number of boxes considered by the algorithm as shown on the right figure. Indeed, boxes that are completely inside  $\mathbb{S}$  no longer need to be considered again.

**Largest negative invariant set** We can compute easily the largest negative invariant associated to the Van Der Pol system, with the same algorithm, by using Remark 2.28:  $\text{Inv}_{\mathbf{f}}^-(\mathbb{X}) = \text{Inv}_{-\mathbf{f}}^+(\mathbb{X})$ . The result is shown in Figure 3.19.

### 3.3.5 Parameters that affect the speed of convergence

We will deal here with the parameters that have been found to have an effect on the speed of convergence, *i.e.* the speed of the decrease of the uncertainty layer. We also give for each of these parameters some perspectives that would improve the computations.

---

<sup>6</sup>All computations in this document, unless otherwise stated, have been conducted on an i5-3320M processor with 8GiB of RAM.

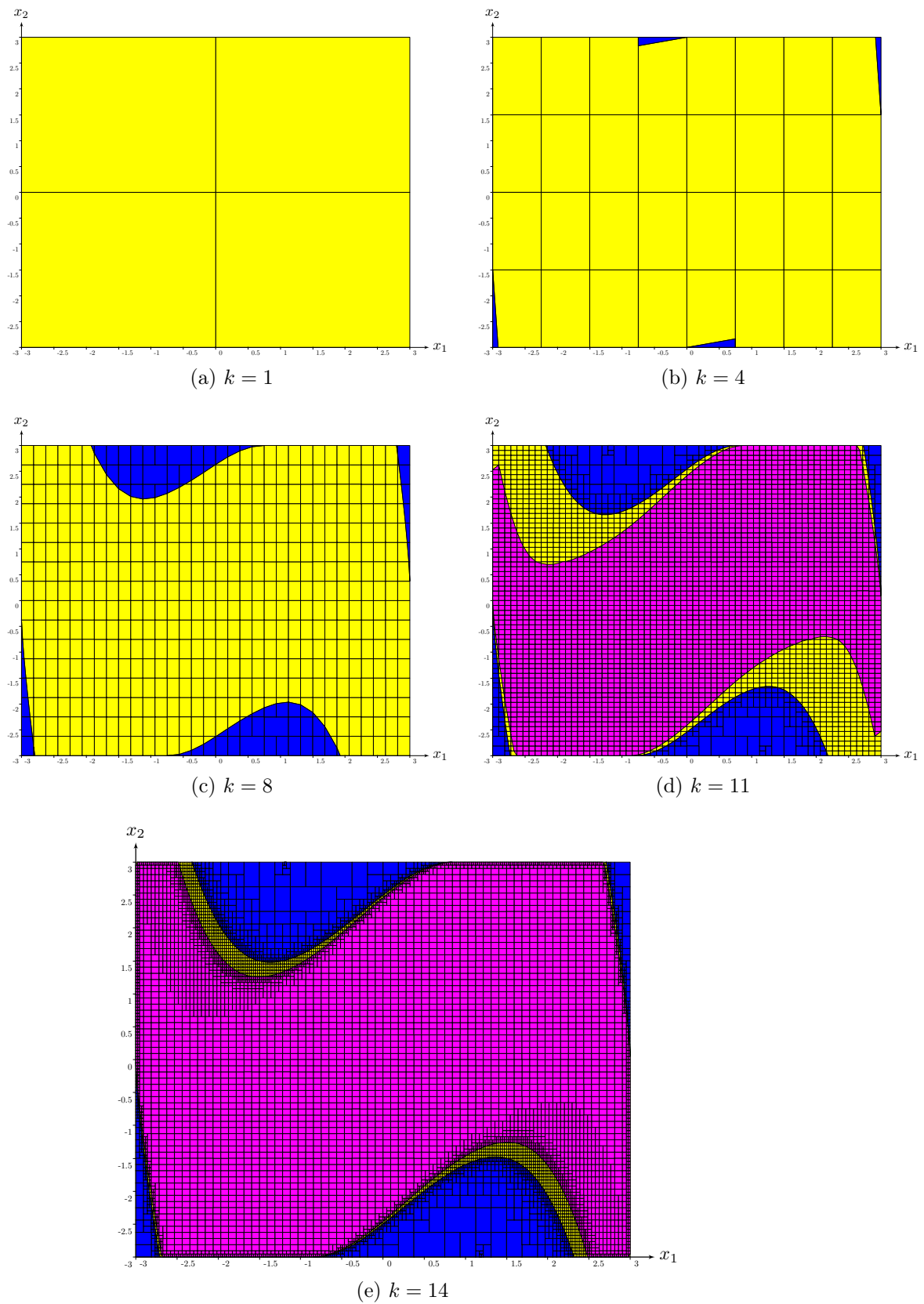
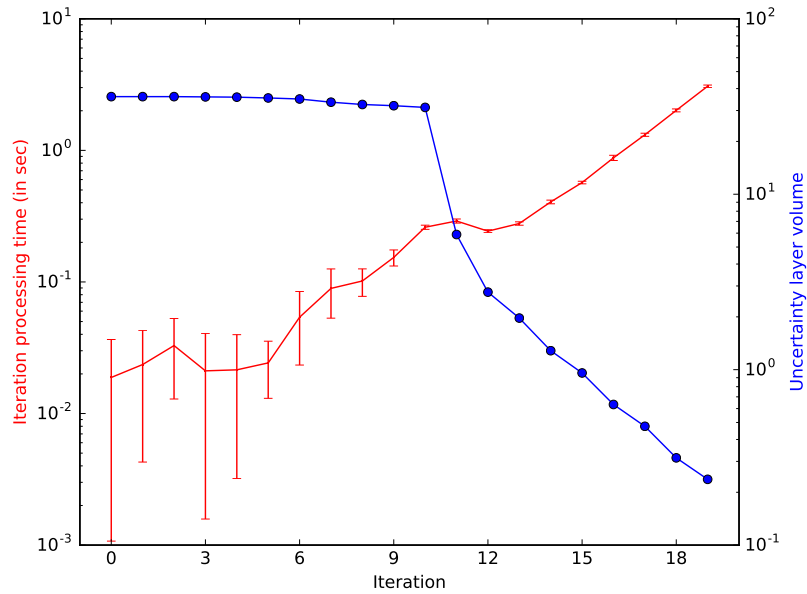
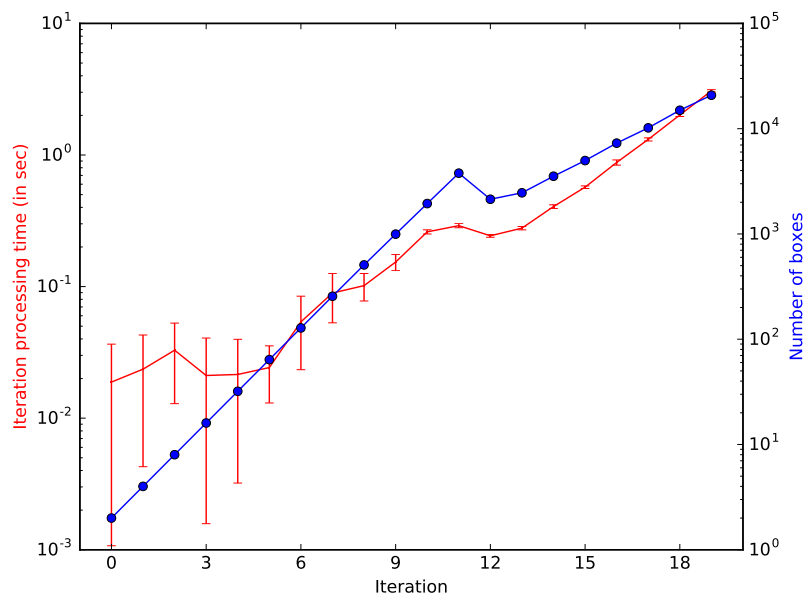


Figure 3.17: Bracket of  $\text{Inv}^+(\mathbf{f}, \mathbb{X})$  at different iteration steps.



(a)



(b)

Figure 3.18: Volume of the uncertainty layer (3.18a) and number of boxes considered (3.18b) along with the processing time at each iteration (standard deviation for 30 samples per iterations).

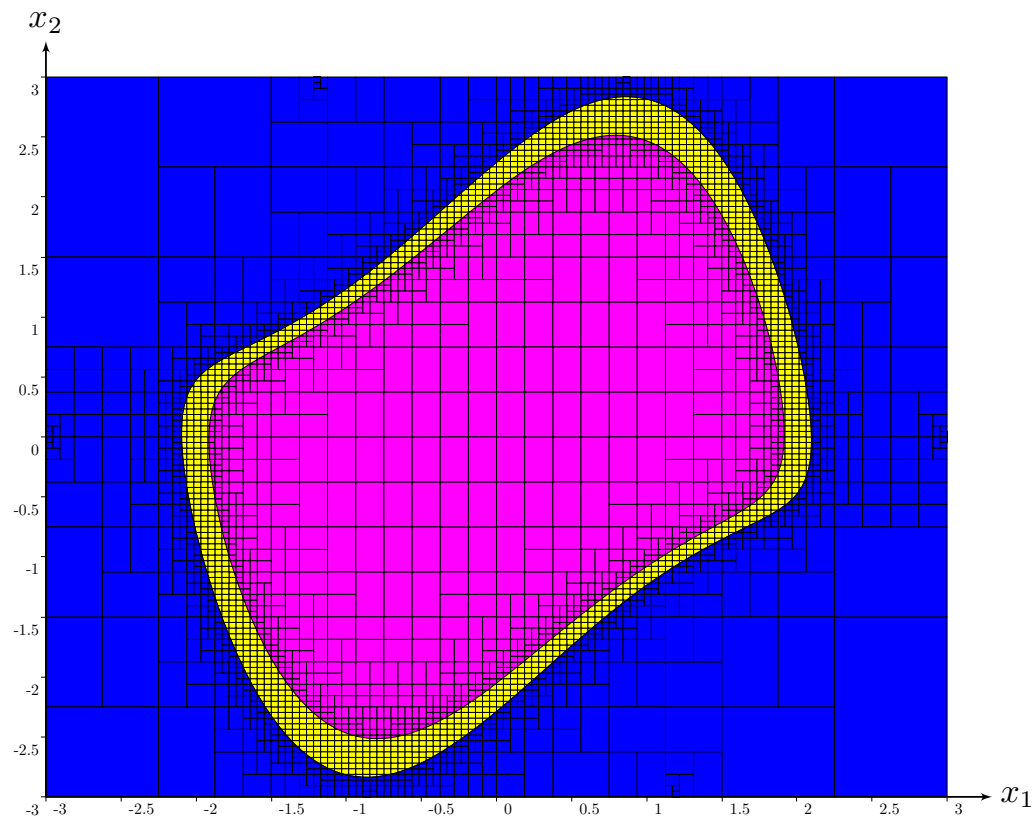


Figure 3.19: Largest negative invariant set of the Van Der Pol System ( $k = 13$ ).

**Vector field approximation** The more the approximation of the vector field in a box is accurate, the better the contraction will be. Providing to the algorithm a *minimal inclusion function* (see Definition 2.76 on page 48) or at least an efficient one reduces the number of bisections needed.

For higher space dimension, the use of at least a first order vector field approximation could improve the approximation. All contractors and inflators would have to be rewritten. The drawback is that the use of a better approximation of the vector field requires more complex computations and costs therefore more time.

**Bisection heuristic** The bisection heuristic is crucial. We could, for instance, focus the effort of bisection to the boxes where the vector field approximation is the less accurate. However, most of the time it is difficult to predict the effect of such heuristic as large areas of the state space, with poor vector field approximation, can be eliminated after the propagation of constraints from an other part of the state space.

Nevertheless, choosing a bisection ratio different than 0.5 or prioritizing the bisection of certain areas of the state space shall be studied in certain specific cases.

**Constraint propagation heuristic** The order in which constraints are resolved has a significant impact on the processing time. This point will be developed later in Section 3.4.3.

**Abstract domain of doors** The choice of the abstract domain of doors is obviously important. The abstract domain should limit as far as possible any *loss of information* but it should also limit too costly computations and too import memory footprint. A compromise need to be found between the two objectives. This point is one of the major limitations of the method to be scaled up to high-dimensional systems.

Choosing different abstract domains for different areas of the state space could be an interesting idea to explore.

**Contractor and inflators** The efficiency of contractor and inflators can be improved. For instance the boundary conditions can be studied at the scale of the door rather than at the box level.

## 3.4 Toward an implementation

In this section, we will focus on advanced properties and issues of mazes that are required to be studied when the method is implemented in a computer. We have chosen to present them in this second stage for the sake of clarity.

### 3.4.1 The sliding issue

The sliding issue is a geometrical problem that appears due to the paving. It degrades the performances of the set approximation using mazes. We present here the phenomenon and we propose a very first approach to handle the issue. A more detailed study shall be undertaken in future works, in particular a general formalization of the problem along with theorems and proofs.

#### 3.4.1.1 Problem statement

Similarly to the problem of infinite time integration with a Lagrangian approach, an issue of infinite transitions occurs with an Eulerian approach. This effect appears when the flow contractor is used. Indeed, some *dead* paths may be kept which produces a larger over approximation than what could have been obtained.

**Example 3.35.** Let us consider the example of Figure 3.20. The initial state of the maze is on the left of the figure and the expected contracted road is on the right. If we use the `FLOWCONTRACTBWD` operator as presented previously, none of the roads are contracted.

Indeed, roads are only considered individually. Indeed, if we consider  $[\mathbf{x}_1]$ , then for all  $\mathbf{a} \in \mathcal{D}(3)$  there exists a ray  $\mathbf{v}$  starting from  $\mathbf{a}$  that does not cross any wall. Indeed, it can escape through the bottom. This is the same if we only consider  $[\mathbf{x}_2]$ . No contraction can therefore be accomplished.

However, from a global point of view, we know that a path starting from  $\mathbf{a}$  will ultimately cross a wall: in our case the left one. It requires here to look at what the path become after living  $[\mathbf{x}_1]$  which means to study the sequence of next transitions. Unfortunately, the path may ultimately cross an infinite number of time the door between  $[\mathbf{x}_1]$  and  $[\mathbf{x}_2]$  before reaching the wall: it is then impossible to study all the transitions.

The phenomenon described by Example 3.35 appears when the path can carry out a sliding between two boxes before leaving them. This can be seen as a Zeno problem. More formally, the sliding phenomenon appears when we



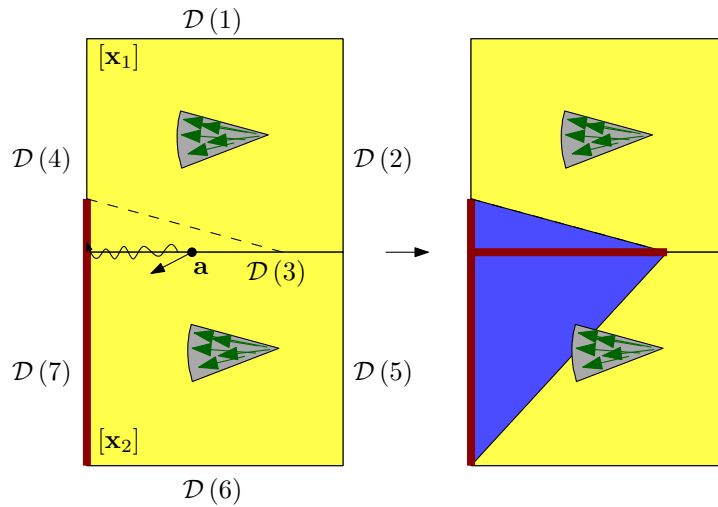


Figure 3.20: A sliding issue example between two 2D boxes.

have for a door:

$$\mathbf{0} \in \{\mathbf{n} \cdot \mathbf{v} \mid \mathbf{v} \in [\mathbf{v}]\} \quad (3.2)$$

where  $\mathbf{n}$  is the normal vector space of the door and  $[\mathbf{v}]$  is the union of all boxes vector fields the door is part of.

**Example 3.36** (Example in higher dimension). In 3D, sliding paths can have a helicoidal pattern. In Figure 3.21 we have represented four boxes associated to a green vector field with two zeros in its component (horizontally). We want to contract the yellow doors according to the red walls. We know that paths, as the orange-painted one, will reach a wall at the end. Similarly to the 2D case, the simple FLOWCONTRACTBWD cannot contract the doors. In this case the path can slide for instance between four boxes with a helicoidal pattern.

### 3.4.1.2 Graph model of Mazes

Mazes can be modeled as a directed graph  $\mathcal{G}$ . The vertexes are the doors and the directional edges correspond to the existence of a possible path between two doors of the same box.

**Example 3.37.** Figure 3.22 represents the directed graph of the maze of Figure 3.20. Each door is represented by a circle. The possible paths between doors, according to the vector flow, are shown through a directional arrow. We have separated the edges in two categories: the blue ones that correspond to paths inside  $[\mathbf{x}_1]$  and the red ones that correspond to paths inside  $[\mathbf{x}_2]$ . The blue and red edges correspond also to a different vector flow.

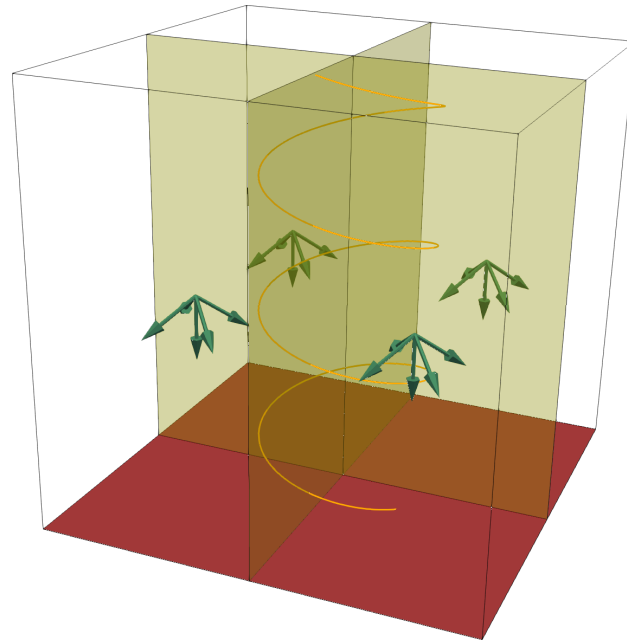


Figure 3.21: Helicoidal sliding trajectories in a 3D maze. The vector field is represented by the set of green arrows. Doors are in yellow and walls in red. An example of path is orange painted.

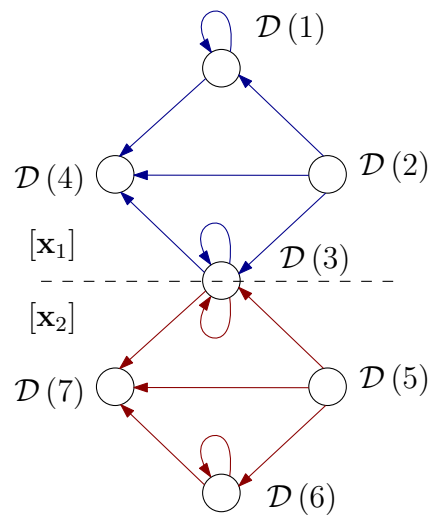


Figure 3.22: Maze and graph  $\mathcal{G}^1$  representation of Figure 3.20.

**Definition 3.38** (Positive neighbors). A door  $\mathcal{D}_2$  is positively linked to a door  $\mathcal{D}_1$  if there exists a path starting from  $\mathcal{D}_1$  that reaches  $\mathcal{D}_2$  without crossing any other doors nor any walls.

We define the set  $\mathbb{N}^+(\mathcal{D})$  of all doors which are positively linked with  $\mathcal{D}$ .  $\mathbb{N}^+(\mathcal{D})$  will be called the positive neighbors of  $\mathcal{D}$ .

**Example 3.39.** In Figure 3.22, we have  $\mathbb{N}^+(\mathcal{D}(3)) = \{\mathcal{D}(4), \mathcal{D}(7), \mathcal{D}(3)\}$ .

*Remark 3.40.* The graph should be carefully interpreted. It is not because  $\mathcal{D}(5)$  is linked to  $\mathcal{D}(3)$  and  $\mathcal{D}(3)$  to  $\mathcal{D}(4)$ , that there will exist a valid path that starts from a point of  $\mathcal{D}(5)$  and reaches  $\mathcal{D}(4)$ . Indeed,  $\mathcal{G}^1$  a *one transition* graph that over approximates the system dynamics. It is analog to an adjacent matrix for a graph. A *two transitions*  $\mathcal{G}^2$  would have given, for instance, the link between two doors if there exists a path that starts from the first one, crosses a door, and reaches the second one without crossing any wall. Algorithm 6, which is used to solve the CN, works only with a *one transition* graph as we study paths from one door to the eventually next door they will cross.

In the general case, some loops that do not exist in the dynamical system appear due to an over approximation of the vector flow. The bisection of the paving usually solves this issue. However, in the sliding case, this will not happen as the loop is due to the door geometrical orientation which is collinear with the vector flow.

**Example 3.41.** In our 2D example a *geometrical loop artifact* appears for  $\mathcal{D}(3)$ . In the 3D case of Figure 3.21, we have the same phenomenon but between four doors and not only a single door.

### 3.4.1.3 Graph rebuilding

To avoid any *loop artifacts*, we will modify locally the graph while keeping all valid paths. This aims to modify the abstract domain to limiting a *loss of information*. The operation can also be seen as a new contractor.

**Example 3.42.** If we take back the example of Figure 3.22, we know that paths will at the end cross  $\mathcal{D}(4)$  or  $\mathcal{D}(7)$ . We can then study how paths starting from  $\mathcal{D}(3)$  will leave both  $[\mathbf{x}_1]$  and  $[\mathbf{x}_2]$  instead of considering them individually. We here adopt a door centered paradigm instead of a box centered paradigm as shown in Figure 3.23. We do not study how the path will evolve between  $[\mathbf{x}_1]$  or  $[\mathbf{x}_2]$  but only if it will leave the union.

More generally, let  $\mathcal{D}_s$  be a door where we have a sliding issue. We want to find a modified set  $\mathbb{N}^+(\mathcal{D}_s)$  where there is no *loop artifacts*. The steps are the following:

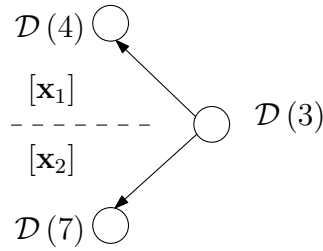


Figure 3.23: Graph rebuilding for the FLOWCONTRACTBWD operator.

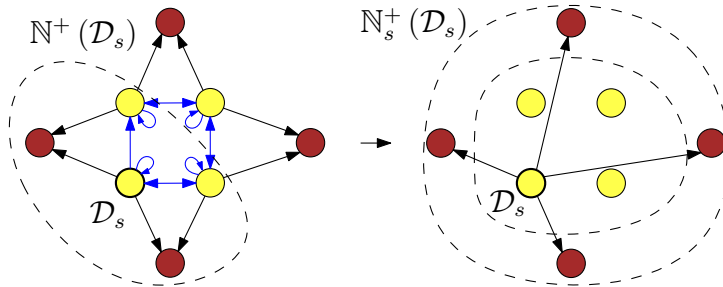


Figure 3.24: Update of the graph neighbors in case of sliding paths for a simplified version of Figure 3.21.

1. We find the set  $\mathbb{C}$  of all doors that are part of directed cycles which includes  $\mathcal{D}_s$ . This set can be easily computed using classic graph algorithms such as the Tarjan's strongly connected components algorithm limited to the strongly connected component that contains  $\mathcal{D}_s$ .
2. We define the new positive neighbors as:

$$\mathbb{N}_s^+(\mathcal{D}_s) = \{\mathcal{D} \in \mathbb{N}^+(\mathcal{D}_C), \mathcal{D}_C \in \mathbb{C} \mid \mathcal{D} \notin \mathbb{C}\}.$$

$\mathbb{N}_s^+(\mathcal{D}_s)$  corresponds to the doors that can be reached from the cycles but that do not belong to the cycles. It means that paths starting from  $\mathcal{D}_s$  will not be trapped into any cycles. This operation can be seen as an operator that modify the graph. We will call it the *graph rebuilder*.

**Example 3.43.** In Figure 3.24, we present a simplified version of the graph of the 3D helicoidal example of Figure 3.21. The red vertexes correspond to the walls (or the closed doors) and the yellow vertexes to the doors. We painted in blue the cycle which contains the door we want to contract. On the right of the figure, we can see the new set  $\mathbb{N}_s^+(\mathcal{D}_s)$  which avoids any *loop artifacts*.

*Remark 3.44.* In some cases, large cycles may appear due to the form of the vector fields. In that case, the *graph rebuilder* has no effect. Indeed, we have

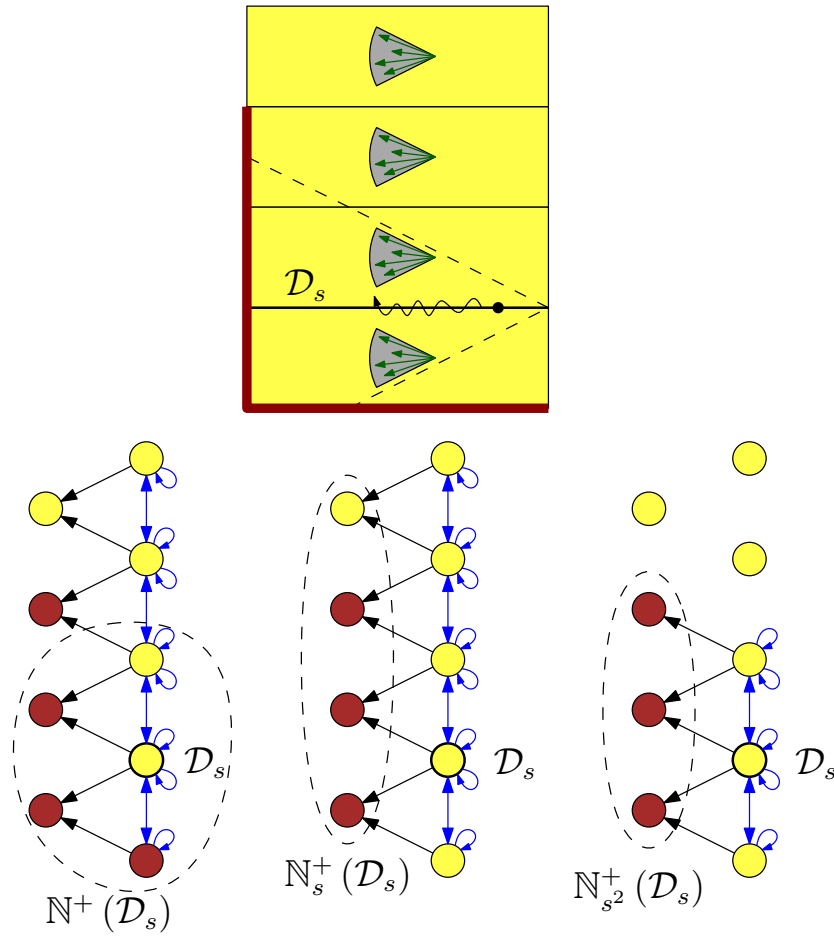


Figure 3.25: A simple example of graph where using the  $\mathcal{G}^2$  to compute the neighbors improves the contractions.

to use the union of all vector fields that can be encountered by paths. In Figure 3.25 we have chosen a very simple and specific example to illustrate a way to deal with large cycles.

There are several ways to improve the algorithm, we can for instance compute the  $\mathcal{G}^2$  for  $\mathcal{D}_s$  to remove some vertexes that cannot be reached by a cycle starting from  $\mathcal{D}_s$ . In the bottom of Figure 3.25, we have represented the following sets:  $\mathbb{N}^+(\mathcal{D}_s)$ ,  $\mathbb{N}_s^+(\mathcal{D}_s)$  and  $\mathbb{N}_{s^2}^+(\mathcal{D}_s)$  where the last one takes into account the  $\mathcal{G}^2$  for  $\mathcal{D}_s$ .

*Remark 3.45.* In our implementation, that is available online (see Section 1.3), we did not fully implement the sliding issue in dimension greater than two. This is why we will mainly focus on 2D examples in the following chapters.

### 3.4.2 Using abstract domains without the ACC

We consider here domains that do not fulfill the ACC such as polytopes. In that case, we have no guarantee that a fixed point can be reached by the abstract domain. This implies that the algorithm will never converge to a fixed point and so will never stop. Unfortunately, we have seen that polytopes were necessary in dimensions greater than two (see Section 3.3.3.3 on page 101).

The problem is slightly different if we deal with the inner or the outer approximation. Indeed, to compute an inner approximation a fixed point must be reached. At the opposite, to compute an outer approximation, we can stop at any time the algorithm as an outer approximation is obtained at every steps<sup>7</sup>.

To tackle the issue in the case of the inner approximation, a *widening operator* has to be used. The idea is to over approximate the inflations in order to overshoot the *lfp*, i.e.  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^\infty$ , and to stop to an over approximation  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^{\infty, \nabla}$ . To improve the result, we can apply a finite number of contractions that will help to obtain a better over approximation of the *lfp*. This will reduce the effect of the widening operator. This operation is called a *narrowing* in the AI community.

**Example 3.46.** Figure 3.26 illustrates the principle. The red arrow corresponds to the inflation of the maze up to an overshoot of the *lfp* ( $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^\infty$ ). We recall that the *lfp* may not be representable, in a computer, by the maze. We therefore have reached an other *lfp* under widening ( $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^{\infty, \nabla} \supseteq \mathcal{L}_{\mathbb{S}^+, \mathcal{P}}^\infty$ ). The blue arrow corresponds to a finite number of contractions that reduce thereafter the uncertainty using the `FLOWCONTRACTBWD` operator. The complementary of the obtained maze is then taken to obtain an inner approximation of  $\mathbb{S}$ .

### 3.4.3 Constraint propagation heuristic and multi-threading capabilities

**Propagation heuristic** The number of roads to process depends mainly on the heuristic of constraint propagation. By heuristic we mean to decide, given a set of roads to process, how we schedule the computation. In the case where we process roads one by one, we can define a set of optimal heuristics that reaches the fixed point in a minimum number of operations. Finding

---

<sup>7</sup>In practice, we allow for each road a maximum number of contractions, when using polytopes.

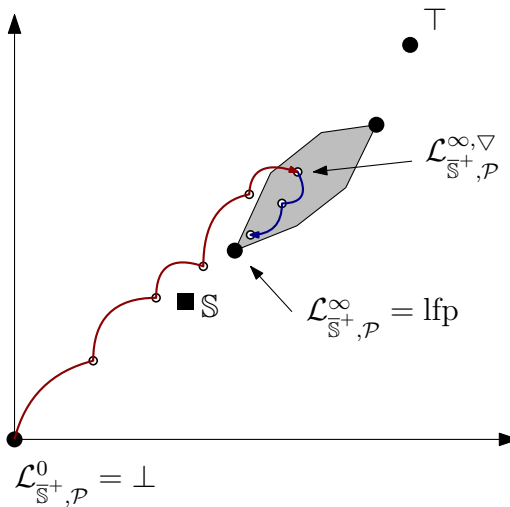


Figure 3.26: Computing the inner approximation with a widening operator. The grey area corresponds to the fixed points of the operator without widening.

these optimal heuristics seems however difficult as it depends mainly on the shape of the vector field.

**Example 3.47.** To illustrate the choice of a heuristic, let us consider the simple example of Figure 3.27. We have been using two different heuristics: one for the top mazes and one for the bottom mazes. In the top case, we start by contracting  $(3) \rightarrow (2) \rightarrow (5) \rightarrow (4)$ , then  $(6) \rightarrow (5) \rightarrow (4) \rightarrow (8) \rightarrow (7)$  and finally  $(9) \rightarrow (8) \rightarrow (7)$ . In the bottom case, we use a different order which is more efficient:  $(3) \rightarrow (6) \rightarrow (9)$ , then  $(2) \rightarrow (5) \rightarrow (8)$  and finally  $(4) \rightarrow (7)$ . In both cases we reached the fixed point. However, in the first case, 12 operations were needed whereas only 8 in the second case. In fact, the second case has here an optimal heuristic where we chose to process boxes “perpendicularly” to the vector field.

**Link to multithreading** One of the benefits of the graph maze structure is that the constraint resolution for each door or road can be multithreaded. We can update in parallel all the doors to a new state according to the previous state of their neighbors. Using multithreading not only increases the number of parallel processing capabilities, but it also helps to better find an optimal heuristic. In the example of Figure 3.27, the optimal heuristic is here sequential: parallel computation is not needed. However, as it is difficult to guess the optimal order, exploring different heuristics simultaneously will improve the computation time.

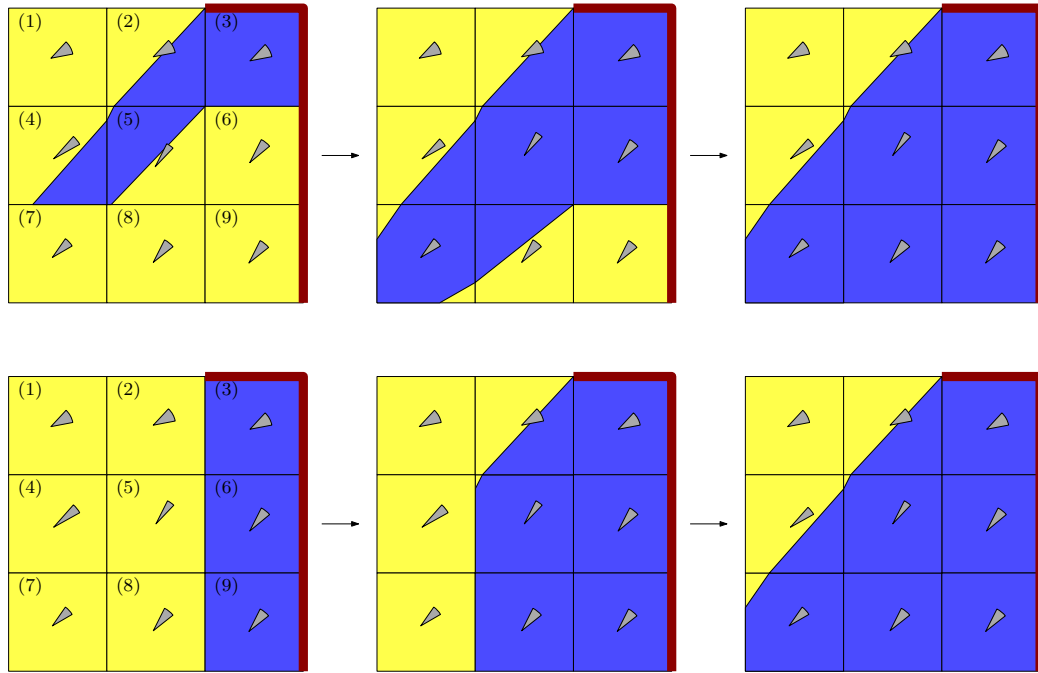


Figure 3.27: Propagation of constraint in a maze using two different heuristics.

**Example 3.48.** Figure 3.28 shows the result of the computation time of the Van Der Pol example using from one to four threads. We clearly see the performance gain between a unique thread and two threads that might be explained by the heuristic of constraint propagation.

### 3.4.4 Using existing libraries

In this section, we will look at the implementation of the `FLOWCONTRACTBWD` and `FLOWINFLATEBWD` operators.

In the 2D case, we can use the visibility contractor (see Section 2.3.2.3) to implement the `FLOWCONTRACTBWD` and `FLOWINFLATEBWD` operators.

In a more nD general case, we will discuss here how the Parma Polyhedra Library (PPL) can be used to implement both operators. We have chosen in PPL the convex closed polytopes to be the abstract domains of doors.

PPL defines and implements a *time-elapse operator* (Halbwachs, Proy, and Roumanoff, 1997) denoted  $\nearrow$  which for two non necessarily closed (NNC) polytopes  $\mathbf{P}, \mathbf{Q}$  in  $\mathbb{R}^n$ , computes  $\mathbf{P} \nearrow \mathbf{Q}$  as the smallest NNC polytope containing the set:

$$\{\mathbf{p} + \lambda \mathbf{q} \in \mathbb{R}^n \mid \mathbf{p} \in \mathbf{P}, \mathbf{q} \in \mathbf{Q}, \lambda \in \mathbb{R}^+\}.$$



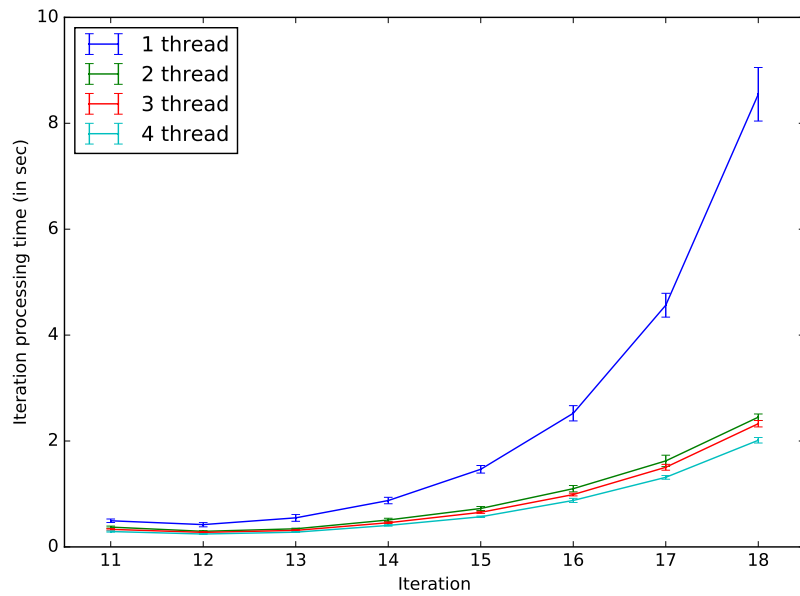


Figure 3.28: Multithreading on the Van Der Pol example of Section 3.3.4 (standard deviation for 30 samples). The difference between one thread and two threads could be explained by a better heuristic of constraint propagation.

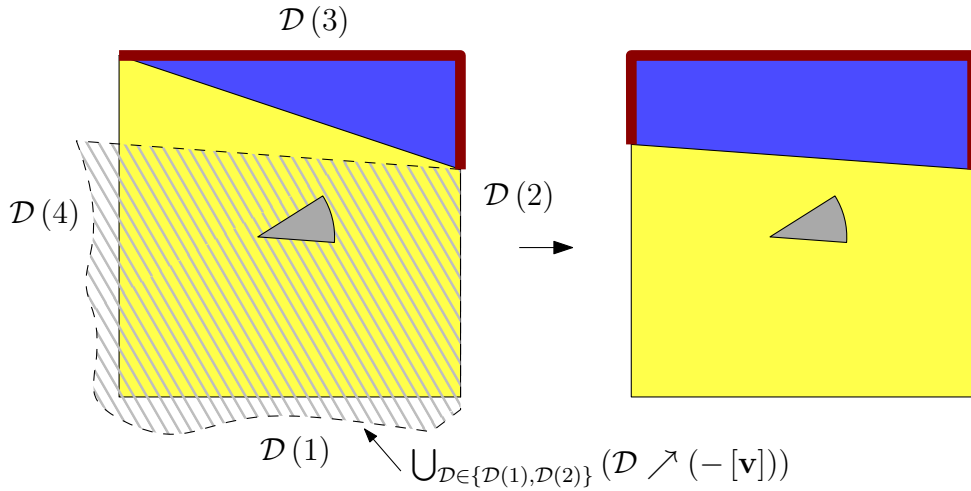


Figure 3.29: The time-elapse operator with the FLOWCONTRACTBWD operator.

Let us take  $\langle [\mathbf{x}], \mathbb{D} \rangle$ , a road composed of a set of doors  $\{\mathcal{D}(0), \dots, \mathcal{D}(l)\}$ . We denote  $\mathcal{D}^\top(\cdot)$  the largest possible state of a door, *i.e.* a completely open door. We set  $[\mathbf{v}]$  as the outer approximation<sup>8</sup> of the vector field in  $[\mathbf{x}]$ . We recall that the intersection of a closed polytope with a NNC polytope is a closed polytope.

The FLOWCONTRACTBWD can be written for each door as:

$$\mathcal{D}(k) = \mathcal{D}(k) \cap \bigcup_{\mathcal{D} \in \mathcal{N}^+(\mathcal{D}(k))} (\mathcal{D} \nearrow (-[\mathbf{v}])). \quad (3.3)$$

The FLOWINFLATEBWD can similarly be written as:

$$\mathcal{D}(k) = \mathcal{D}(k) \cup \bigcup_{\mathcal{D} \in \mathcal{N}^+(\mathcal{D}(k))} ((\mathcal{D} \nearrow (-[\mathbf{v}])) \cap \mathcal{D}^\top(k)). \quad (3.4)$$

**Example 3.49.** Figure 3.29 illustrates equation (3.3) with a 2D example. The door  $\mathcal{D}(4)$  is contracted using the FLOWCONTRACTBWD operator.

*Remark 3.50.* In the case of sliding paths, we use the modified version of the neighbors  $\mathcal{N}_s^+(\mathcal{D}(k))$  for the FLOWCONTRACTBWD. The vector field  $[\mathbf{v}]$  is changed to the union of all the boxes vector fields the path can cross. This is not necessary for the FLOWINFLATEBWD.

We can note that the implementation of the FLOWCONTRACTBWD and FLOWINFLATEBWD are relatively closed. This also justifies the use of a complementary approach to compute the inner approximation (see Section 3.3.2).

<sup>8</sup> $[\mathbf{v}]$  is a box and so a particular type of polytope.

## 3.5 Extension to differential inclusion and system with input

In this section, we will focus on extending the previous results about bracketing a *largest positive invariant set* to the bracket of a *largest positive viable set*. The dynamical system will be defined by a differential inclusion or an evolution function with an input. The set  $\mathbb{S}$  is now defined as  $\mathbb{S} = \text{Viab}_f^+(\mathbb{X}, \mathcal{U})$ , and we will try to compute an outer  $\mathbb{S}^+$  and an inner  $\mathbb{S}^-$  approximation of  $\mathbb{S}$ .

The dynamical system will be modeled by the following state equation:

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

where  $\mathbf{x} \in \mathbb{R}^n$  is the state vector,  $\mathbf{f}$  the evolution function and  $\mathbf{u} \in \mathcal{U}$  an input function or a noise function.

We will show that mazes and their dedicated algorithms, presented previously, can be directly extended to compute the outer approximation and require only some adjustments to compute the inner approximation.

*Remark 3.51.* Definition 3.1 of *valid* paths can be extended to the case where the system has an input. An initial condition is considered *valid* or *viable* if it is associated to an input function  $\mathbf{u} \in \mathcal{U}$  such that the path is *feasible* and all its points belong to  $\mathbb{X}$ .

### 3.5.1 Outer approximation

We recall, that to compute an outer approximation  $\mathcal{L}_{\mathbb{S}^+, \mathcal{P}}$  for a given paving  $\mathcal{P}$ , we use a contractor approach by *removing dead paths without removing any valid paths* (see Section 3.3.1 on page 89). This means that a state  $\mathbf{x} \in \mathbb{R}^n$  is outside  $\mathbb{S}^+$  if:  $\forall \mathbf{u} \in \mathcal{U}, \exists t \geq 0, \varphi_{\mathbf{f}, \mathbf{u}}(\mathbf{x}, t) \notin \mathbb{X}$ . We then must update the  $\text{FLOWCONTRACTBWD}_f(\mathcal{L})$  but we can keep the  $\text{BOUNDARYCONTRACT}_{\mathbb{X}}(\mathcal{L})$ .

The update constraint is the following: doors are contracted without loosing any point in the set of *all*  $\mathbf{a} \in \text{DOORS}([\mathbf{x}])$  such that there exists a ray starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [\mathbf{f}]([\mathbf{x}], [\mathbf{u}])$  that does not intersect  $\text{WALLS}([\mathbf{x}])$ , where  $[\mathbf{u}] = [\{\mathbf{u}(\mathbf{x}) \mid \forall \mathbf{x} \in [\mathbf{x}], \forall \mathbf{u} \in \mathcal{U}\}]$ .

*Remark 3.52.* The only difference in the algorithm is the way the over approximation of the vector field is computed. We use the box hull of the union of all vector fields associated to all the possible inputs.

**Example 3.53.** Figure 3.30 shows, similarly to Figure 3.9, the  $\text{FLOWCONTRACTBWD}_f(\mathcal{L})$  version with an input. We have drawn the vector field for two examples of input (in green):  $\mathbf{u}_1$  and  $\mathbf{u}_2$  which both belong

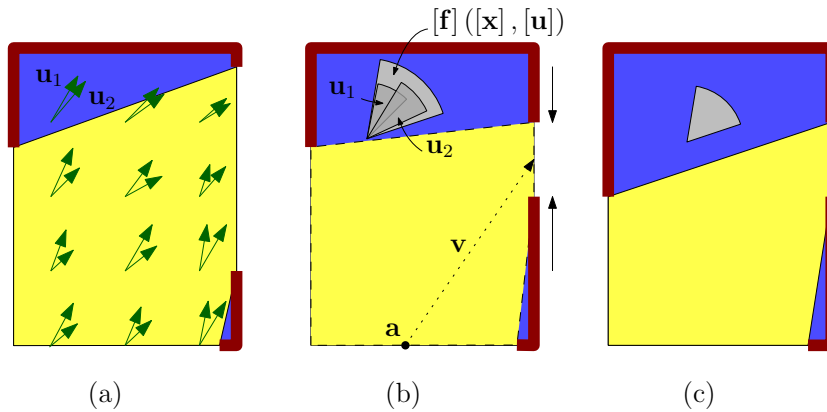


Figure 3.30: The flow contractor with an input.

to  $\mathcal{U}$ . The large gray cone corresponds to  $[f]([\mathbf{x}], [\mathbf{u}])$  and we can see that  $[f]([\mathbf{x}], \mathbf{u}_1)$  and  $[f]([\mathbf{x}], \mathbf{u}_2)$  are subsets of this cone. Doors will be contracted for the states  $\mathbf{x}$  if for any input  $\mathbf{u} \in \mathcal{U}$ , the path reaches a wall.

### 3.5.2 Inner approximation

The inner approximation is more difficult to obtain. The principle is still to add *all dead paths* to the maze. We keep the complementary approach and the idea of an inflation of the doors with a  $\text{FLOWINFLATEBWD}_f(\mathcal{L})$  operator.

The new constraint is the following: we inflate the doors to enclose all points of the set  $\mathbb{A}$  of *all*  $\mathbf{a} \in \text{WALLS}([\mathbf{x}])$  such that for all  $\mathbf{u} \in \mathcal{U}$ , there exists a ray starting from  $\mathbf{a}$  of direction  $\mathbf{v} \in [f]([\mathbf{x}], \mathbf{u})$  that intersects  $\text{DOORS}([\mathbf{x}])$ .

**Example 3.54.** To illustrate the inflator, let us consider the simple example in Figure 3.31. We have a box where the set  $\mathcal{U}$  is composed of only two constant input functions  $\mathbf{u}_1$  and  $\mathbf{u}_2$ , *i.e.*  $\mathcal{U} = \{\mathbf{u}_1, \mathbf{u}_2\}$ . Similarly to Figure 3.12, the right door is inflated after an external event (middle maze). For the state  $\mathbf{a}$ , we can choose the input  $\mathbf{u}_1$  in order to be sure to not cross a door. This is the same for the state  $\mathbf{c}$  but with the input  $\mathbf{u}_2$ . Both states should not then be added at that stage to  $\mathcal{L}_{\overline{\mathbb{S}}^+, \mathcal{P}}$ . This is not the case for the state  $\mathbf{b}$  as we cannot find an input  $\mathbf{u}$  such that there does not exist a ray  $\mathbf{v} \in [f]([\mathbf{x}], \mathbf{u})$  that might intersect a door. Indeed, we do not know if the path will finally cross the door but there is a *risk* due to the over-approximation of the vector field.

From the previous example, we can see that computing the set  $\mathbb{A}$  amounts to compute the intersection for all  $\mathbf{u} \in \mathcal{U}$  of the  $\text{FLOWINFLATEBWD}_f$  oper-

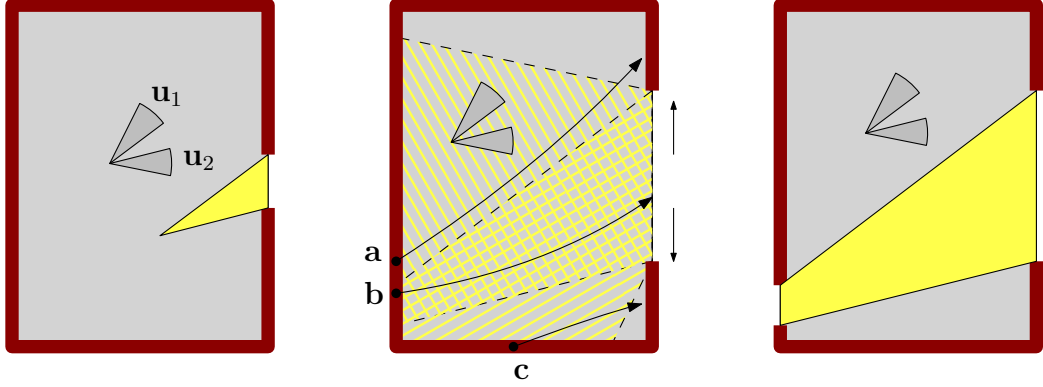


Figure 3.31: Illustration of the flow inflate with a set of two input functions.

ator. Indeed, we have:

$$\begin{aligned}
 & \left\{ \mathbf{a} \in \text{WALLS}([\mathbf{x}]) \mid \forall \mathbf{u} \in \mathcal{U}, \exists \mathbf{v} \in [\mathbf{f}](\mathbf{x}, \mathbf{u}), \forall t \geq 0, \right. \\
 & \quad \left. \varphi_{\mathbf{f}, \mathbf{u}}(\mathbf{a}, t) \in \text{DOORS}([\mathbf{x}]) \right\} \\
 &= \bigcap_{\mathbf{u} \in \mathcal{U}} \left\{ \mathbf{a} \in \text{WALLS}([\mathbf{x}]) \mid \exists \mathbf{v} \in [\mathbf{f}](\mathbf{x}, \mathbf{u}), \forall t \geq 0, \right. \\
 & \quad \left. \varphi_{\mathbf{f}, \mathbf{u}}(\mathbf{a}, t) \in \text{DOORS}([\mathbf{x}]) \right\} \\
 &= \bigcap_{\mathbf{u} \in \mathcal{U}} \{ \text{FLOWINFLATEBWD}_{\mathbf{f}}(\text{WALLS}([\mathbf{x}])) \}
 \end{aligned}$$

*Remark 3.55.* If we compute the previous intersection for a subset of  $\mathcal{U}$  instead of for all  $\mathbf{u} \in \mathcal{U}$ , we will obtain an over approximation of the set that will give an under approximation of  $\mathbb{S}^-$ . This point is important as it justifies that we can choose a finite set of input functions to compute the inner approximation.<sup>9</sup> This is indeed required to be able to implement the algorithm. Note that for the outer approximation the whole set  $\mathcal{U}$  is considered as we take its convex hull.

In practice, we will choose to define  $\mathcal{U}$  as the set of piecewise functions that have a constant value  $\{\mathbf{u}_1, \dots, \mathbf{u}_l\}$  for each box. The more the number of possible constant values will be taken into account in each box, the more the inner approximation will be accurate. To be efficient, each constant value should be chosen in order to provide a boundary behavior: for example a maximum left or right turn for a car.

*Remark 3.56.* To avoid any issues with sliding paths, we shall verify that

<sup>9</sup>For instance the edges of the convex hull of  $\mathcal{U}$

there is no zero component in the union of all the inputs. The test condition of Equation 3.2 becomes:

$$\mathbf{0} \in \{\mathbf{n} \cdot \mathbf{v} \mid \mathbf{v} \in [\mathbf{f}]([\mathbf{x}], [\mathbf{u}])\}.$$

*Remark 3.57.* Similarly to Section 3.4.4, the `FLOWINFLATEBWDf` is implemented using the PPL with the following equation:

$$\mathcal{D}(k) = \mathcal{D}(k) \cup \bigcap_{\mathbf{u} \in \{\mathbf{u}_1, \dots, \mathbf{u}_l\}} \left( \bigcup_{\mathcal{D} \in \mathbb{N}^+(\mathcal{D}(k))} ((\mathcal{D} \nearrow (-[\mathbf{u}])) \cap \mathcal{D}^\top(k)) \right). \quad (3.5)$$

Note that the union operator, which is usually a convex hull operator, produces an over-approximation and thus should be applied as late as possible. We can improve Equation 3.5 through a simple re-factorizing:

$$(\mathcal{D}_{\mathbf{u}_1,1} \cup \mathcal{D}_{\mathbf{u}_1,2} \cup \dots) \cap (\mathcal{D}_{\mathbf{u}_2,1} \cup \mathcal{D}_{\mathbf{u}_2,2} \cup \dots) \cap \dots = (\mathcal{D}_{\mathbf{u}_1,1} \cap \mathcal{D}_{\mathbf{u}_2,1} \cap \dots) \cup (\mathcal{D}_{\mathbf{u}_1,1} \cap \mathcal{D}_{\mathbf{u}_2,2} \cap \dots) \cup \dots$$

We then have:

$$\mathcal{D}(k) = \mathcal{D}(k) \cup \bigcup_{\{\mathcal{D}_1, \dots, \mathcal{D}_l \mid \mathcal{D}_1, \dots, \mathcal{D}_l \in \mathbb{N}^+(\mathcal{D}(k))\}} \left( \bigcap_{\langle \mathbf{u}, \mathcal{D} \rangle \in \{\langle \mathbf{u}_1, \mathcal{D}_1 \rangle, \dots, \langle \mathbf{u}_l, \mathcal{D}_l \rangle\}} ((\mathcal{D} \nearrow (-[\mathbf{u}])) \cap \mathcal{D}^\top(k)) \right). \quad (3.6)$$

However, more operations will be generated with Equation 3.6 as the number of operations will be exponential with the number of  $\mathbf{u}_l$  considered.

### 3.5.3 The example of the reverse Van Der Pol system with an input

To illustrate this extension to dynamical systems with an input, we will consider a reverse Van Der Pol system with an input. This system has a stable equilibrium point in  $\mathbf{0}$  and an unstable cycle. We choose the following evolution function:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = - \begin{pmatrix} x_2 + u_1 \\ (1 - x_1^2)x_2 - x_1 + u_2 \end{pmatrix}$$

where  $\mathbf{u} \in \mathcal{U} = [-0.2, 0.2] \times [-0.5, 0.5]$ . We want to compute  $\text{Viab}_f^+(\mathbb{X}, \mathcal{U})$  for  $\mathbb{X} = [-3, 3] \times [4, 4] \setminus \{\mathbf{x} \mid x_1^2 + x_2^2 \leq 1\}$ . It means that we remove the

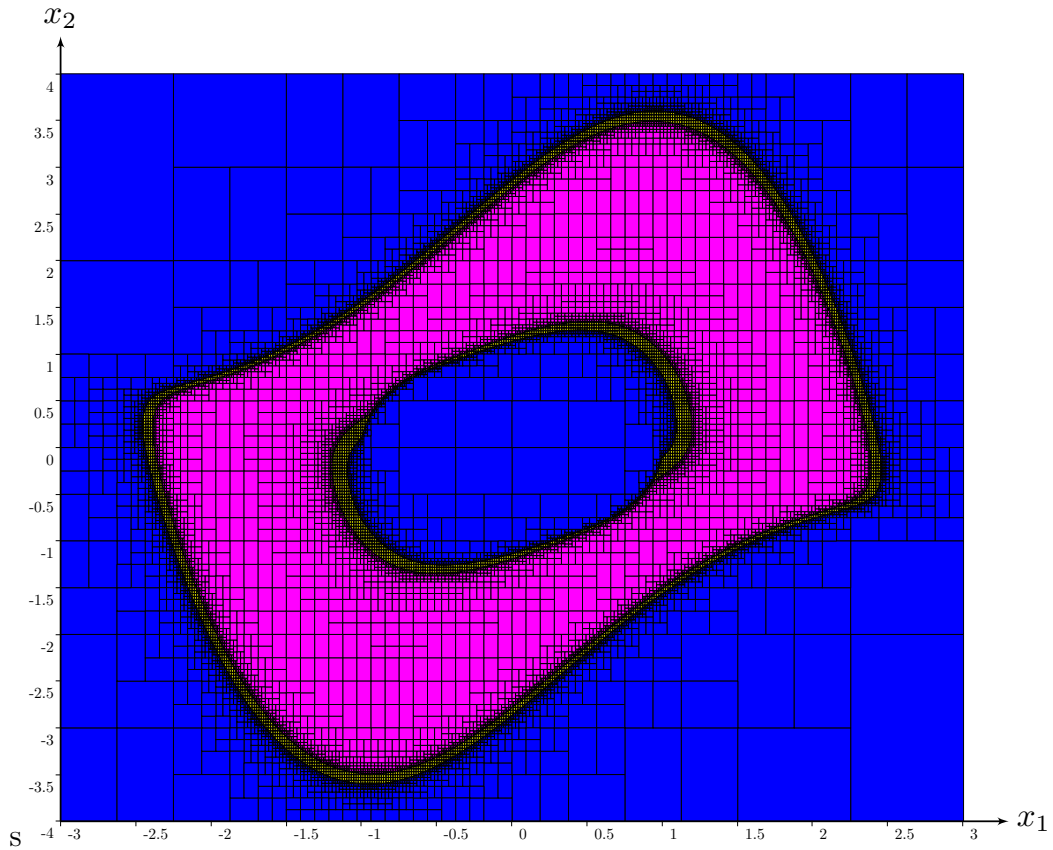


Figure 3.32: Example of the largest viable set for the reverse Van Der Pol system with an input.

stable equilibrium point. Therefore, the system is unstable for a constant input function.

We chose four different inputs to compute the inner approximation that are the edges of the convex hull of  $\mathcal{U}$ :  $\mathbf{u}_1 = (-0.2, -0.5)^\top$ ,  $\mathbf{u}_2 = (-0.2, 0.5)^\top$ ,  $\mathbf{u}_3 = (0.2, -0.5)^\top$ ,  $\mathbf{u}_4 = (0.2, 0.5)^\top$ . Figure 3.32 shows the result of the algorithm.

## 3.6 Conclusion

In this chapter, we have been focusing on bracketing the largest positive invariant sets of a non linear dynamical system. The problem has been formalized through the CN framework. We have proposed a new domain called *maze* that abstracts paths. We have also presented tools such as inflators and contractors that enable to take into account constraints on

paths. Finally, a general algorithm was proposed to obtain the bracket of the largest positive invariant.

In a second time we have highlighted the limits of *mazes* and we have shown how the algorithm could be implemented in a computer. The main drawback of *mazes* is that the associated algorithms are exponential with the dimension of the state space. However, several interesting results were obtained on different examples such as the bracket of the largest invariant set of a Van Der Pol system. Moreover, there are several perspectives of improvement of the algorithms in particular concerning the sliding issue. This would probably allow to deal efficiently with problems that have a dimension greater than two.

Finally, we have shown how we can extend, with little changes, the algorithms to compute the largest viable set of a dynamical system.

We now have an efficient tool to compute positive or negative invariant sets. We will see in the next chapter how this tool can be the cornerstone to solve more complex problems.



# Chapter 4

## Applications of invariant sets

### Contents

---

<b>4.1</b>	<b>The largest positive and negative invariant sets</b>	<b>. 126</b>
4.1.1	The problem	126
4.1.2	Application: the example of isobath navigation	127
<b>4.2</b>	<b>Forward &amp; Backward reach sets</b>	<b>. 134</b>
<b>4.3</b>	<b>Attraction basin</b>	<b>. 141</b>
<b>4.4</b>	<b>Capture reach set</b>	<b>. 145</b>
<b>4.5</b>	<b>Eulerian state estimation</b>	<b>. 147</b>
4.5.1	Formalism	149
4.5.2	Invariant sets approach	149
<b>4.6</b>	<b>Conclusion</b>	<b>. 156</b>

---

In this chapter we will show how several classic problems involving dynamical systems can be translated into the bracket of an invariant set. We will successively look at the problem of bracketing the largest invariant sets, *i.e.* positively and negatively, bracketing backward and forward reach sets, bracketing attraction basins and bracketing capture reach sets. Finally, we will propose an Eulerian state estimator that synthesizes the previous problems in a unique framework.

We will try to give throughout this chapter, practical examples that illustrate how invariant sets can be used to solve robotic problems.

## 4.1 The largest positive and negative invariant sets

### 4.1.1 The problem

A positive and negative invariant set can be computed from the positive invariant set and the negative invariant set. Indeed, we have  $\text{Inv}_{\mathbf{f}}(\mathbb{X}) = \text{Inv}_{\mathbf{f}}^+(\mathbb{X}) \cap \text{Inv}_{\mathbf{f}}^-(\mathbb{X})$  as proved in Remark 2.28. We also recall that at each step of the Algorithm 6, we have with  $\mathbb{S}_p = \text{Inv}_{\mathbf{f}}^+(\mathbb{X})$  and  $\mathbb{S}_n = \text{Inv}_{\mathbf{f}}^-(\mathbb{X})$ :

$$\begin{cases} \overline{\mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty} \subseteq \text{Inv}_{\mathbf{f}}^+(\mathbb{X}) \subseteq \mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty \\ \overline{\mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty} \subseteq \text{Inv}_{\mathbf{f}}^-(\mathbb{X}) \subseteq \mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty \end{cases} .$$

As mazes and invariant sets have a lattice structure, we have<sup>1</sup>:

$$\left( \overline{\mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty} \cap \overline{\mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty} \right) \subseteq \text{Inv}_{\mathbf{f}}(\mathbb{X}) \subseteq \left( \mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty \cap \mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty \right)$$

which is equivalent by applying the De Morgan's law<sup>2</sup> to:

$$\left( \overline{\mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty} \sqcup \overline{\mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty} \right) \subseteq \text{Inv}_{\mathbf{f}}(\mathbb{X}) \subseteq \left( \mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty \cap \mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty \right) .$$

We can then compute at each step  $k$  the set  $\text{Inv}_{\mathbf{f}}(\mathbb{X})$ . This result can be extended similarly to the *largest viable set*  $\text{Viab}_{\mathbf{f}}(\mathbb{X}, \mathcal{U})$ .

*Remark 4.1.* In practice, for the outer approximation, the intersection between  $\mathcal{L}_{\mathbb{S}_p^+, \mathcal{P}(k)}^\infty$  and  $\mathcal{L}_{\mathbb{S}_n^+, \mathcal{P}(k)}^\infty$  can be computed during the constraint propagation process by applying successively  $\text{FLOWINFLATEBWD}_{\mathbf{f}}(\cdot)$  and  $\text{FLOWINFLATEBWD}_{-\mathbf{f}}(\cdot)$  to the doors. However, in the case of the inner

<sup>1</sup>see Remark 3.17 on the intersection of two mazes with the same subpaving

<sup>2</sup>The law is recalled in Remark 2.16 on page 23

approximation, it is more efficient to delay the union operation after the fix point as the *join* operator adds uncertainties.

**Example 4.2.** Let us consider the following system which models the magnetic field of a dipole:

$$\begin{cases} x_1 = \frac{x_1+1}{(x_1+1)^2+x_2^2} - \frac{(x_1-1)}{(x_1-1)^2+x_2^2} \\ x_2 = \frac{x_2}{(x_1+1)^2+x_2^2} - \frac{x_2}{(x_1-1)^2+x_2^2} \end{cases}$$

and let  $\mathbb{X} = [-2, 2] \times [-2, 2]$ . We want to compute the set  $\text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ ,  $\text{Inv}_{\mathbf{f}}^-(\mathbb{X})$  and  $\text{Inv}_{\mathbf{f}}(\mathbb{X})$ . The system has an attractive equilibrium point in  $\mathbf{x}_a = (1, 0)^\top$  and a repulsive one in  $\mathbf{x}_r = (-1, 0)^\top$ .

Figure 4.1 shows the largest positive invariant  $\text{Inv}_{\mathbf{f}}^+(\mathbb{X})$ , the largest negative invariant  $\text{Inv}_{\mathbf{f}}^-(\mathbb{X})$  and the vector field of the system. For these sets, we use results from Chapter 3. Figure 4.2 shows the largest positive and negative invariant  $\text{Inv}_{\mathbf{f}}(\mathbb{X})$ . We see that in this example, the three sets are different from each other.

### 4.1.2 Application: the example of isobath navigation

In this section, we will show, by way of example, how we can use invariant sets to guarantee the controller of a robot. The example is not directly linked to the problem of ocean currents navigation, but we thought that it is an interesting illustration of invariants.

**The problem** We consider here the problem of isobath navigation from (Jaulin, 2018). A low-cost AUV should navigate in a known bathymetry<sup>3</sup> map  $h(x, y)$  such as the one presented in the introduction (see Figure 1.3 on page 8). The robot can be modeled using the following state equations:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} x \\ y \\ \psi \end{pmatrix} = \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \\ u \end{pmatrix}$$

The robot is assumed to be able to compute the local value of  $h$  by adding its depth  $z$  and its altitude  $z - h(x, y)$  which is measured by an acoustic echo sounder (see Figure 4.3). We also assume that it can measure the gradient

---

<sup>3</sup>an underwater Digital Elevation Model

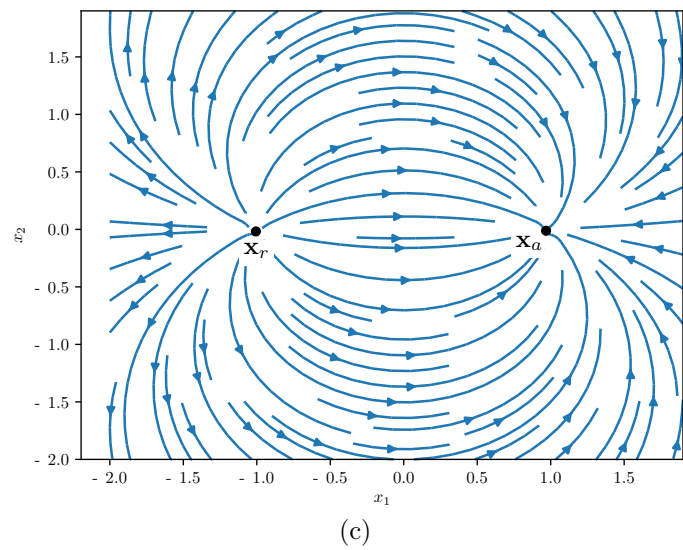
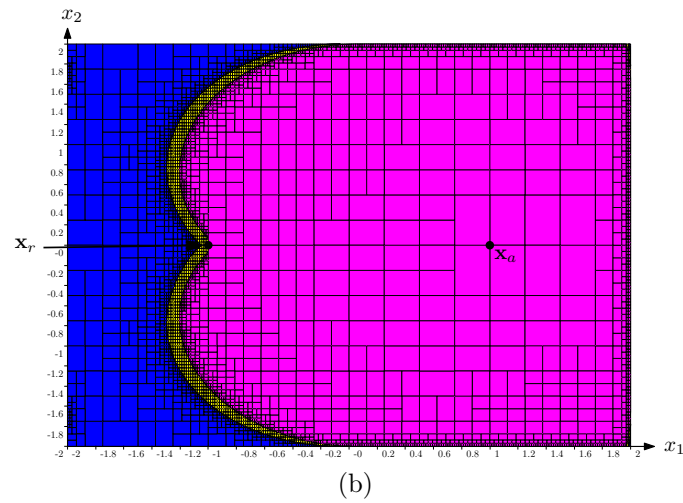
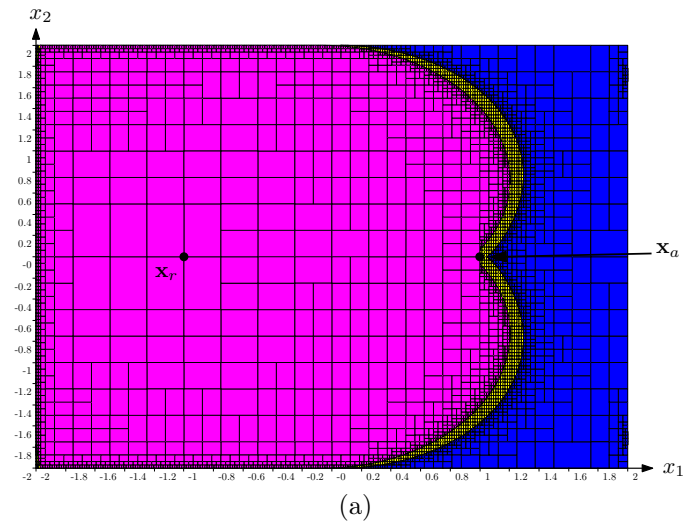


Figure 4.1: The largest negative invariant (a), the largest positive invariant (b) for the dipole system and the associate vector field (c).

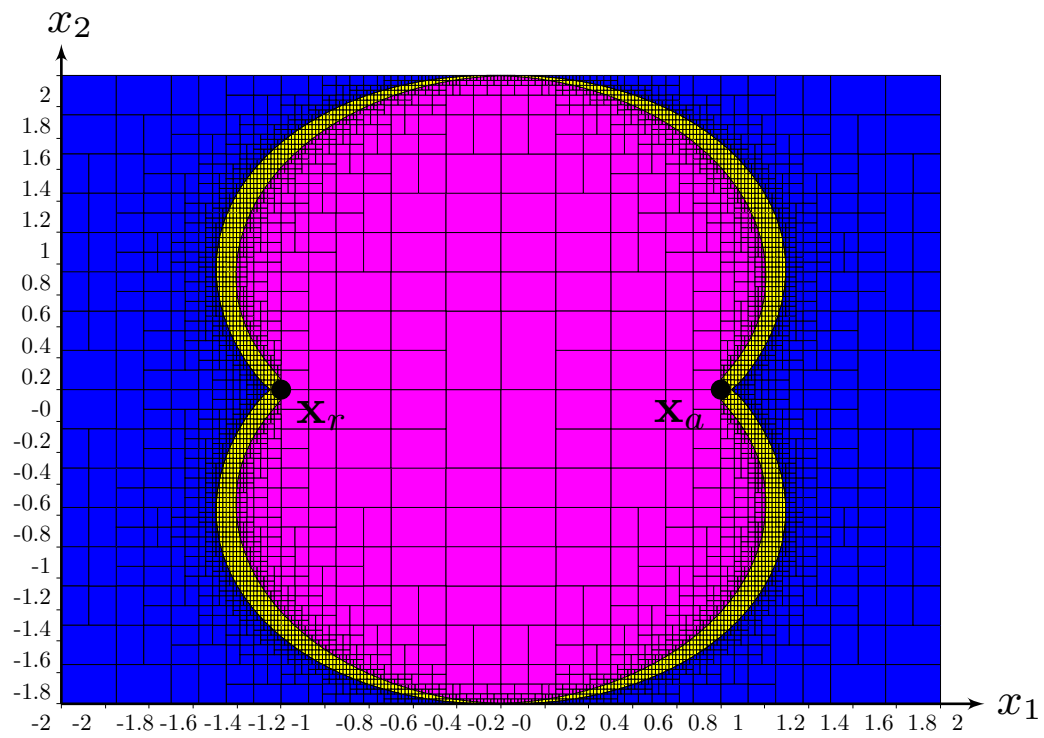


Figure 4.2: The largest positive and negative invariant for the dipole system.

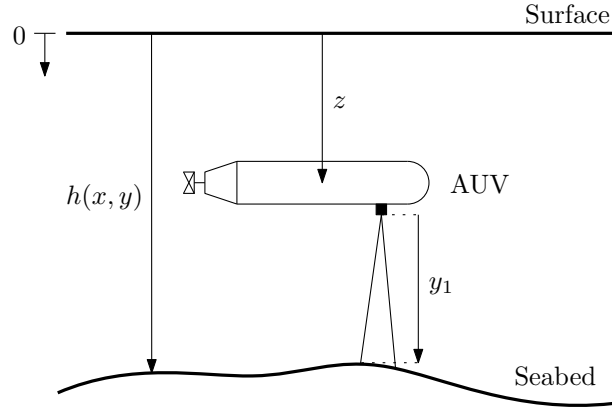


Figure 4.3: Isobath navigation with an AUV.

of  $h$  in its frame.<sup>4</sup> The observation function of the system is:

$$\begin{cases} y_1 = h(x, y) - z \\ y_2 = \text{angle}(\nabla h(x, y)) - \psi, \\ y_3 = z \end{cases}$$

where  $y_1$  is the altitude,  $y_2$  is the gradient of  $h$  in the robot frame and  $z_3$  the depth.

As it is difficult to localize robots underwater, a navigation strategy that consists of following isobaths, *i.e.* underwater level curves, is proposed. For instance, the robot can be set to follow 10 m isobaths with the controller proposed in (Jaulin, 2018):

$$u = -\tanh(h_0 + y_3 + y_1) + \text{sawtooth}\left(y_2 + \frac{\pi}{2}\right),$$

where  $h_0$  is the desired depth of the isobath and  $\text{sawtooth}(\theta) = 2 \arctan\left(\tan\left(\frac{\theta}{2}\right)\right)$ .

We then have a closed-loop system of the form  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ . We assume that the robot is released in an area  $\mathbb{X} \subset \mathbb{R}^2$ . Two questions can be raised:

1. Can we find the set of positions where we can release the robot such that it will stay forever inside  $\mathbb{X}$ ?
2. Can we find the set of paths toward which the robot will converge if it stays forever inside  $\mathbb{X}$ ?

<sup>4</sup>In practice, this data can be retrieved using a Kalman filter and the movements of the robot.

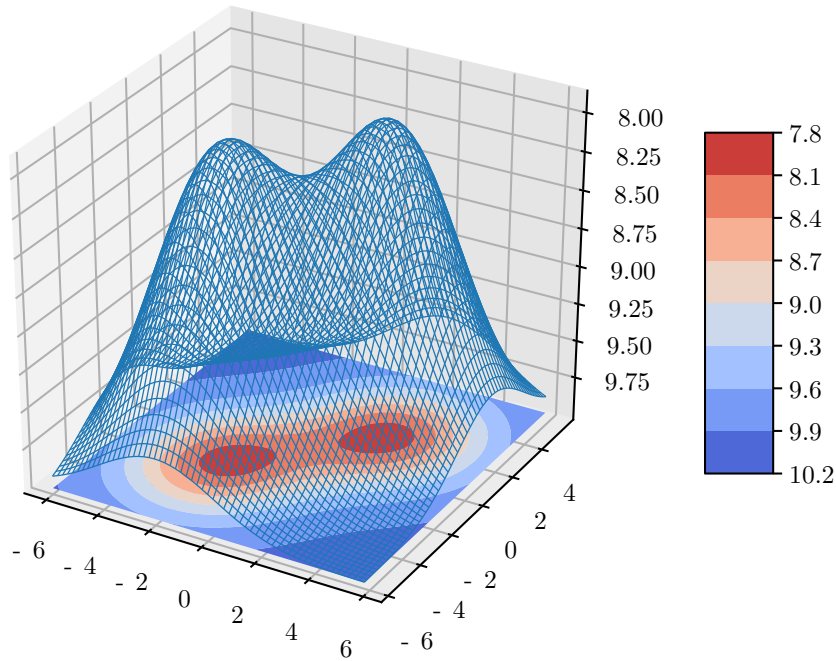


Figure 4.4: Bathymetry of the example.

The first problem amounts to bracket the largest positive invariant set  $\text{Inv}_f^+(\mathbb{X})$  of the system. The second problem amounts to bracket the largest positive and negative invariant set  $\text{Inv}_f(\mathbb{X})$ . Indeed,  $\text{Inv}_f(\mathbb{X})$  combines the paths that will stay forever in the future in  $\mathbb{X}$  and in this set, we only keep paths that were forever in the past inside  $\mathbb{X}$ . Therefore, if the robot is released in  $\text{Inv}_f^+(\mathbb{X})$ , it will converge toward  $\text{Inv}_f(\mathbb{X})$  as we are in a 2D system (see the Poincaré–Bendixson Theorem 2.13 on page 21).

**Example** Let us take the following bathymetry function (see Figure 4.4):

$$h(x, y) = 2e^{-\frac{(x+2)^2+(y+2)^2}{10}} + 2e^{-\frac{(x-2)^2+(y-2)^2}{10}} - 10,$$

and we set  $\mathbb{X} = [-6, 6] \times [-6, 6] \setminus \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$  where  $\mathbb{C}_1, \mathbb{C}_2$  and  $\mathbb{C}_3$  are three disks of radius 0.5, centered respectively in  $(1.78129, 1.78129)^\top$ ,  $(-1.78129, -1.78129)^\top$  and  $(0, 0)^\top$ . They correspond to the two summits and the saddle point.

Figure 4.4 shows  $\text{Inv}_f^+(\mathbb{X})$  and  $\text{Inv}_f(\mathbb{X})$  if we set  $h_0 = 9$  m. Note that  $\text{Inv}_f(\mathbb{X})$  does not have an inner approximation as the set has no volume.

*Remark 4.3.* In the case of a real bathymetry dataset, we can use the dataset contractor<sup>5</sup>,  $\mathcal{C}_{\text{dataset}}$ , to efficiently evaluate the interval version of  $h$  by con-

<sup>5</sup>see Section 2.3.3.2 and in particular Example 2.120

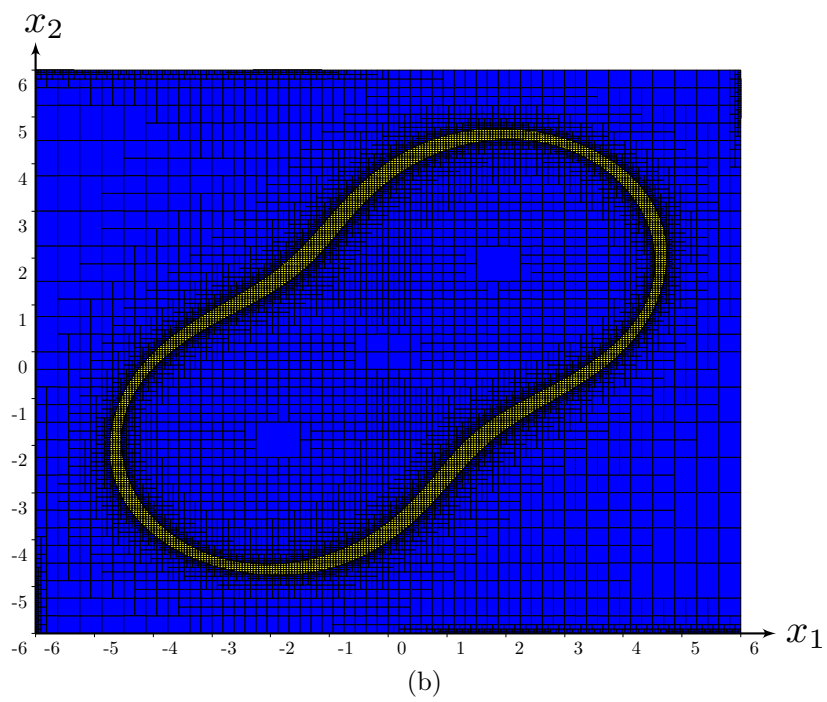
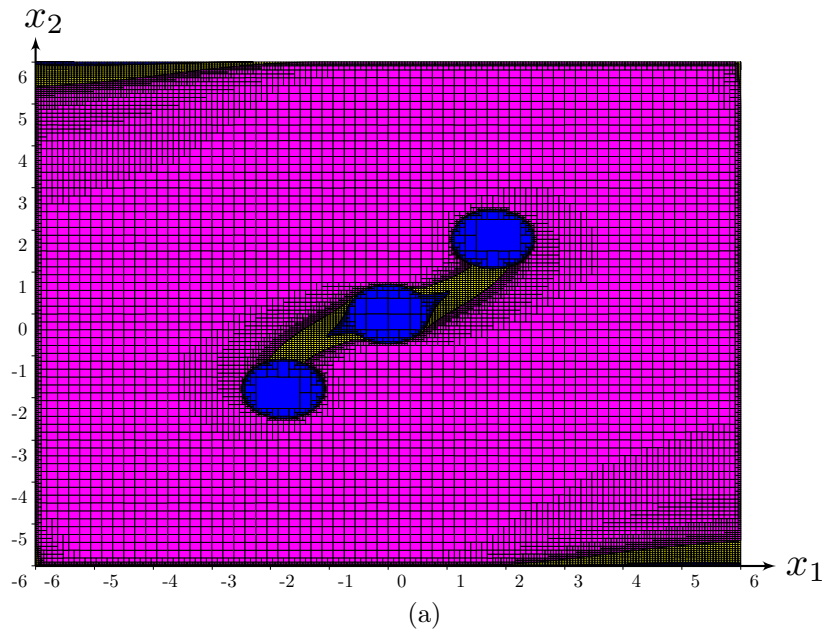


Figure 4.5: Bracket of (a) the  $\text{Inv}_f^+(\mathbb{X})$ , and of (b) the  $\text{Inv}_f(\mathbb{X})$  for the isobath navigation problem.



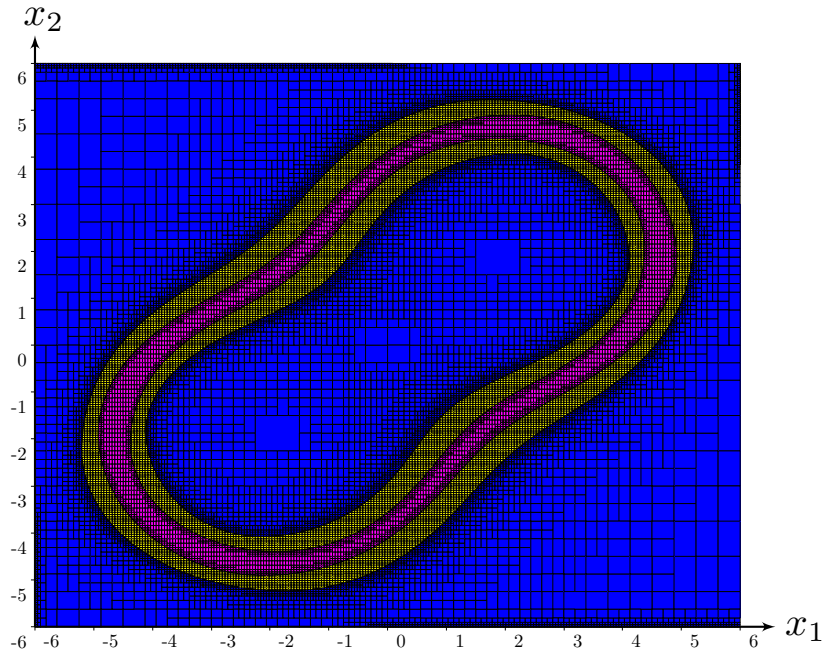


Figure 4.6: Bracketing of  $\text{Viab}_f(\mathbb{X}, \mathcal{W})$  for the isobath navigation problem with measure uncertainties.

tracting the box  $[\mathbf{x}] = [x] \times [y] \times [z]$  where  $[z]$  is the bathymetry. Here we have a space of dimension  $\mathbb{R}^{2 \times 1}$ . The contractor can be used with an initial  $[z]$  value equal to  $[-\infty, \infty]$ . The interval  $[z]$  will be then contracted to an outer approximation of the bathymetry in the area  $[x] \times [y]$ .

**Taking into account uncertainties** We can improve the model used to describe the robot by adding uncertainties on the sensors. If we assume that the altitude is measured with some uncertainties that are bounded inside an interval  $[w]$ , we obtain:

$$y_1 = h(x, y) - z + w, w \in [w].$$

We can similarly bracket the set of Figure 4.4 but using here  $\text{Viab}_f(\mathbb{X}, \mathcal{W})$  where the input is the noise. Figure 4.6 shows the result. The result should be interpreted as follows: the robot will not be able to reach, in the worst case, a set of position thinner than the bracketed set.

## 4.2 Forward & Backward reach sets

Computing a forward or backward reach set can be formalized as bracketing the solution set of an *initial value problem* (IVP) with uncertain initial conditions. The problem can be written as finding the paths that are solution of the following constraints:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{x}_0 \in \mathbb{T} \end{cases} .$$

Solving an IVP can either help to simulate the system or could guarantee a proof of safety. Indeed, it can ensure that the system will never reach, for instance, unwanted states.

There exists a vast literature that focus on this subject with associated solvers such as *CAPD*<sup>6</sup>, the *Tubex* library<sup>7</sup> and the ones cited in Section 1.1.3. Two main approaches exist (Mitchell, 2007): a Lagrangian and Eulerian one. These methods were generally restricted to linear dynamics (Girard, Le Guernic, and Maler, 2006) and have been extended recently to nonlinear systems (Asarin, Dang, and Girard, 2003; Asarin, Dang, and Girard, 2007).

Lagrangian approaches mainly rely on guaranteed integration (Sandretto and Chapoutot, 2016; Wilczak and Zgliczyński, 2011; Rohou et al., 2019). As shown in (Lhommeau, Jaulin, and Hardouin, 2011) a Lagrangian method requires however many bisections with respect to the time line (for the integration of the state equation), but also in the state space.

On their side, Eulerian approaches can be used with nonlinear systems (Quincampoix, 1992; Gao, Lygeros, and Quincampoix, 2006; Kaynama et al., 2012) and try to avoid the integration of the state equation. Most of the corresponding algorithms rely on a gridding of the state space (Saint-Pierre, 2002). To provide guaranteed results, gridding methods require the knowledge of some Lipschitz constants which are rarely available in practice (Saint-Pierre, 1994). Lyapunov-based methods (Ratschan and She, 2010; Delanoue, Jaulin, and Cottenceanu, 2006; Gonzaga, Jungers, and Daafouz, 2012; Barreiro, Aracil, and Pagano, 2002), level-set methods (Mitchell, Bayen, and Tomlin, 2001; Biemond and Michiels, 2014), or barrier functions (Bouissou et al., 2014; Esterhuizen and Lévine, 2016) can also be considered as Eulerian since they only check the constraints on the state space and do not need to perform any integration through time. Few existing methods compute an inner approximation (Goubault and Putot, 2017).

<sup>6</sup>Computer Assisted Proofs in Dynamics group (<http://capd.ii.uj.edu.pl>)

<sup>7</sup><https://github.com/SimonRohou/tubex-lib>

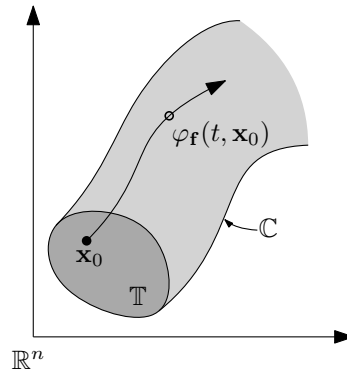


Figure 4.7: Forward reach set  $\mathbb{C}$  of an initial set condition  $\mathbb{T}$ .

We will show how we can rewrite an IVP as a problem of bracketing invariant sets.

**Definition 4.4.** The *forward reach set* of an initial set  $\mathbb{T} \subset \mathbb{R}^n$  is the set of all vectors  $\mathbf{x}$  for which there exists  $\mathbf{x}_0 \in \mathbb{T}$  such that  $\mathcal{S}$  reaches  $\mathbf{x}$  in a finite positive time (see Figure 4.7):

$$\text{Fwd}_{\mathbf{f}}(\mathbb{T}) = \{\mathbf{x} \mid \exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, \mathbf{x} = \varphi_{\mathbf{f}}(t, \mathbf{x}_0)\}$$

The *backward reach set* of a target set  $\mathbb{T} \subset \mathbb{R}^n$  is the set of initial states  $\mathbf{x}_0$  from which  $\mathcal{S}$  reaches the target  $\mathbb{T}$  in a finite negative time:

$$\text{Bwd}_{\mathbf{f}}(\mathbb{T}) = \{\mathbf{x} \mid \exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, \mathbf{x} = \varphi_{\mathbf{f}}(-t, \mathbf{x}_0)\}$$

Bracketing a *forward reach set* or a *backward reach set* can be rewritten as the bracket of a positive invariant set.

**Theorem 4.5.** *We have:*

$$\begin{aligned} \text{Fwd}_{\mathbf{f}}(\mathbb{T}) &= \mathbb{R}^n \setminus \text{Inv}^+(-\mathbf{f}, \mathbb{R}^n \setminus \mathbb{T}), \\ \text{Bwd}_{\mathbf{f}}(\mathbb{T}) &= \mathbb{R}^n \setminus \text{Inv}^-(-\mathbf{f}, \mathbb{R}^n \setminus \mathbb{T}). \end{aligned}$$

*Proof.* Take an element  $\mathbf{x}_a$  of  $\text{Inv}^+(-\mathbf{f}, \mathbb{R}^n \setminus \mathbb{T})$ , we have (see Figure (4.8)):

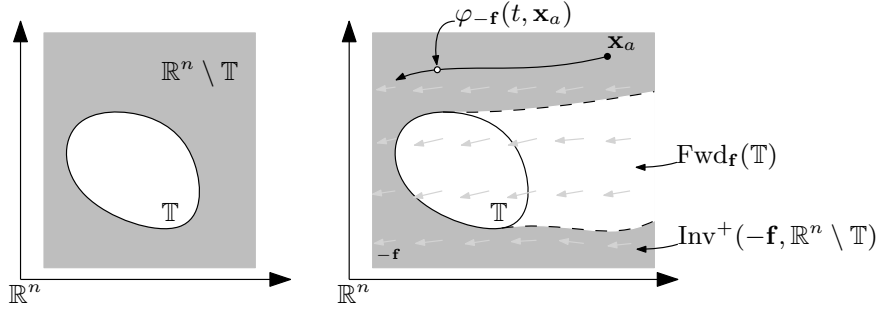


Figure 4.8: Illustration of the proof of Theorem 4.5.

$$\begin{aligned}
\mathbf{x}_a \in \text{Inv}^+(-\mathbf{f}, \mathbb{R}^n \setminus \mathbb{T}) &\Leftrightarrow \varphi_{-\mathbf{f}}([0, \infty], \mathbf{x}_a) \subset \mathbb{R}^n \setminus \mathbb{T} \\
&\Leftrightarrow \forall t \geq 0, \varphi_{-\mathbf{f}}(t, \mathbf{x}_a) \in \mathbb{R}^n \setminus \mathbb{T} \\
&\Leftrightarrow \forall t \geq 0, \varphi_{-\mathbf{f}}(t, \mathbf{x}_a) \notin \mathbb{T} \\
&\Leftrightarrow \neg(\exists t \geq 0, \varphi_{-\mathbf{f}}(t, \mathbf{x}_a) \in \mathbb{T}) \\
&\Leftrightarrow \neg(\exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, \varphi_{-\mathbf{f}}(t, \mathbf{x}_a) = \mathbf{x}_0) \\
&\Leftrightarrow \neg(\exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, \varphi_{\mathbf{f}}(t, \mathbf{x}_0) = \mathbf{x}_a) \\
&\Leftrightarrow \neg(\mathbf{x}_a \in \text{Fwd}_{\mathbf{f}}(\mathbb{T}))
\end{aligned}$$

□

*Remark 4.6.* As for positive and negative invariant sets, *forward* and *backward* reach sets are closely linked:

$$\text{Fwd}_{\mathbf{f}}(\mathbb{T}) = \text{Bwd}_{-\mathbf{f}}(\mathbb{T}).$$

This is directly due to the fact that the forward reach set can be rewrite in terms of invariant sets and because  $\text{Inv}_{\mathbf{f}}^-(\mathbb{X}) = \text{Inv}^+(-\mathbf{f}, \mathbb{X})$ .

**Example 4.7** (Van Der Pol). Let consider the Van Der Pol system with an initial set

$$\mathbb{T} = \{\mathbf{x} \mid (x_1 - 1.2)^2 + (x_2 - 1)^2 \leq (0.3)^2\}.$$

We will bracket  $\text{Fwd}_{\mathbf{f}}(\mathbb{T})$  and  $\text{Bwd}_{\mathbf{f}}(\mathbb{T})$ . Figure 4.9 shows the result of the computation. The set  $\mathbb{T}$  is red painted. The light red boxes around  $\mathbb{T}$  correspond to the outer approximation of  $\mathbb{T}$ .

We can also choose the initial set  $\mathbb{T}$  as a union of sets. For instance, Figure 4.10 shows the  $\text{Fwd}_{\mathbf{f}}(\mathbb{T})$  set for  $\mathbb{T} = [0.6, 1.4] \times [-0.4, 0.4] \cup [-1.0, -0.6] \times [0.3, 0.7]$ .

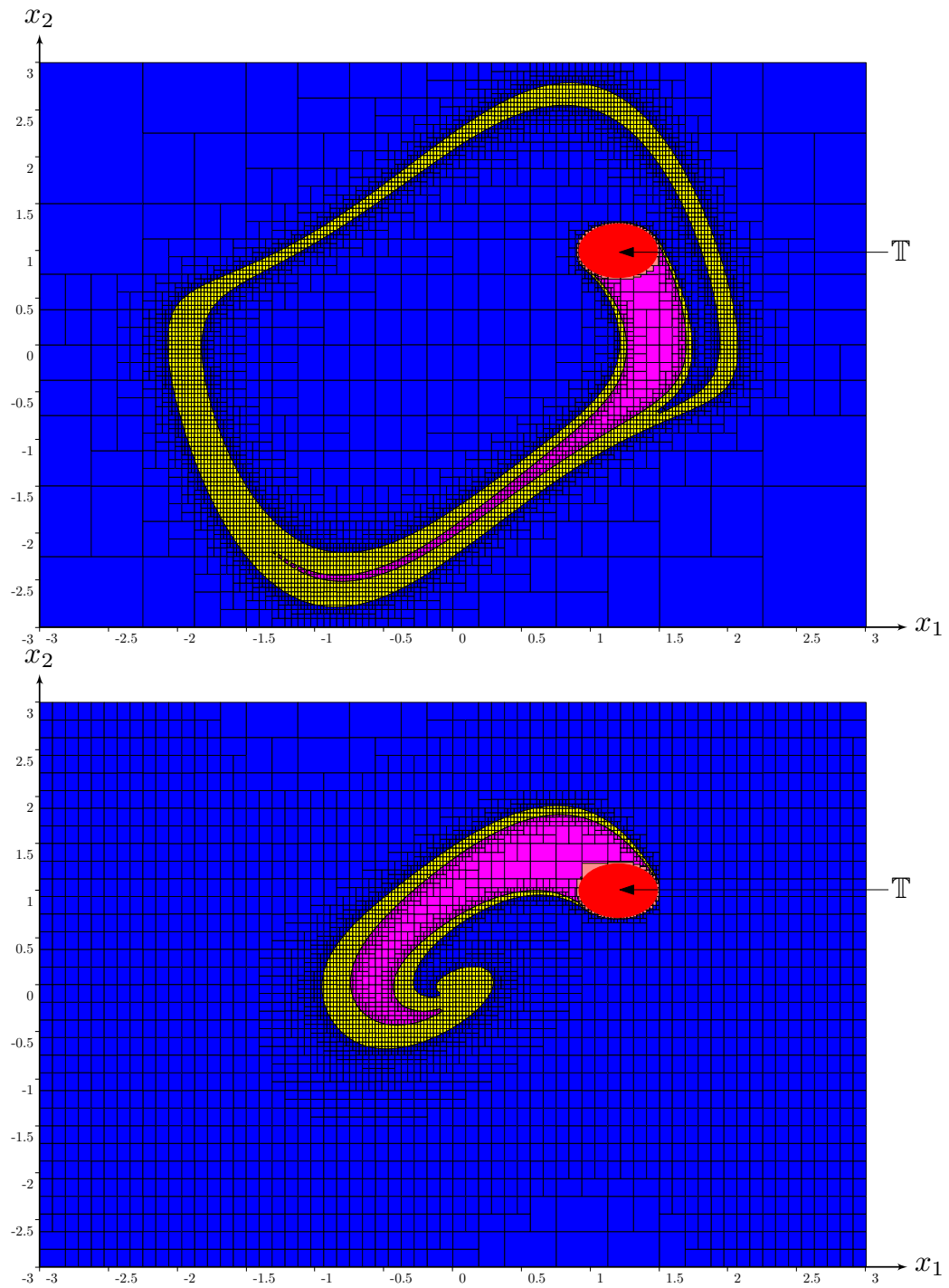


Figure 4.9: Forward and Backward reach set of the Van Der Pol system.

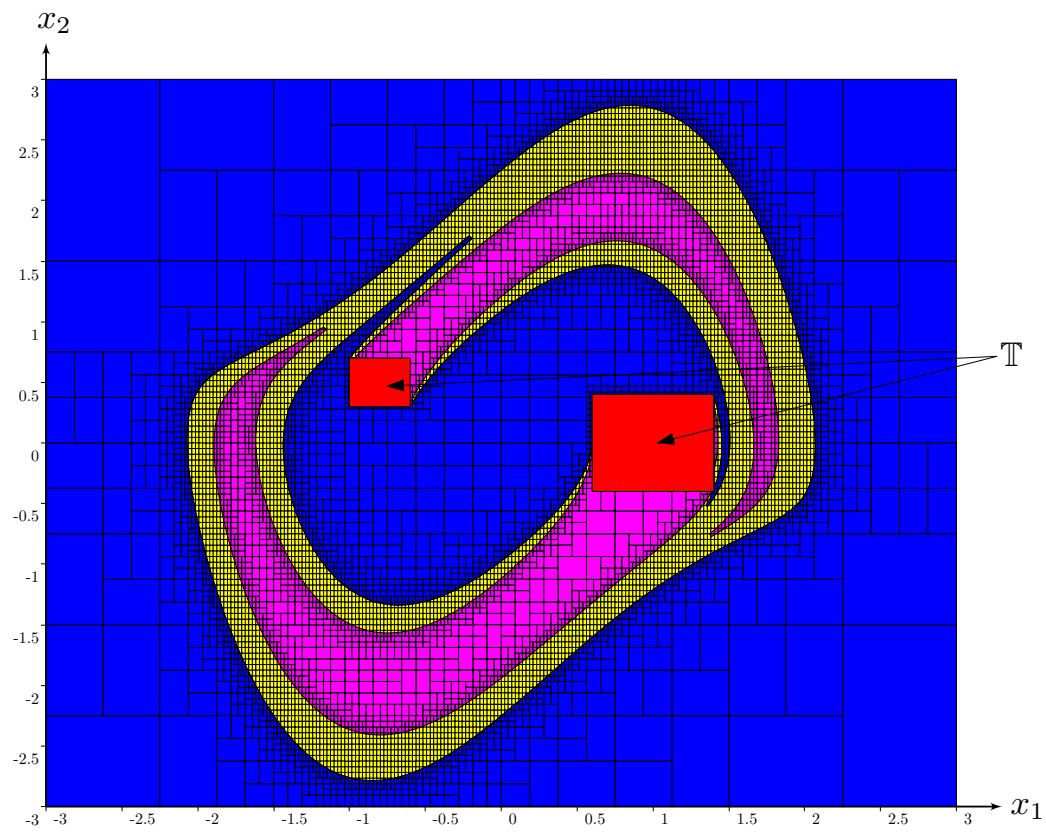


Figure 4.10: Forward reach set from a union of closed sets for the Van Der Pol system.

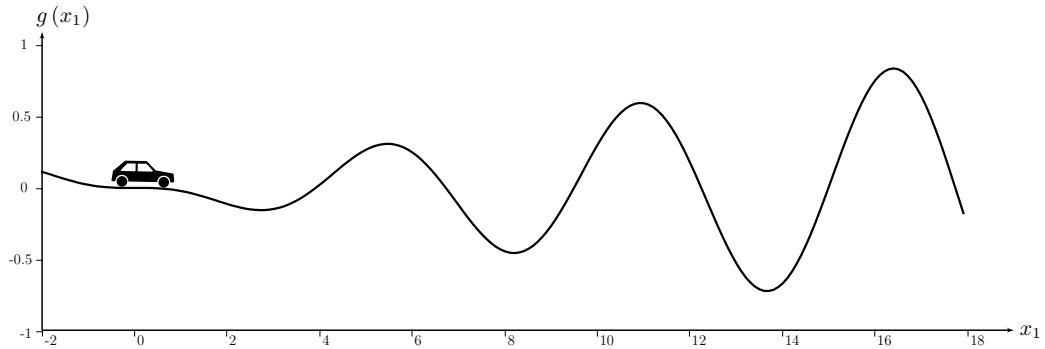


Figure 4.11: Shape of the hill.

*Remark 4.8.* In our implementation of the algorithms described in Chapter 3, a subpaving  $\mathbb{P}$  of the state space was used instead of a paving of the whole space. Indeed, this is easier to implement. The consequence for the forward and backward reach sets is that we only have a guaranteed outer approximation if the outer approximation does not intersect the boundary of the subpaving. Indeed, if there are an intersection of the outer approximation with the boundary, this means that some trajectories can escape and may re-enter inside  $\mathbb{P}$ . A solution to this issue is to consider infinite boxes that covers the whole state space as described in (Le Mézo, Jaulin, and Zerr, 2018).

*Remark 4.9.* We can also extend the forward or backward reach set to systems with inputs:  $\text{Fwd}_{\mathbf{f},\mathcal{U}}(\mathbb{T}) = \mathbb{R}^n \setminus \text{Viab}^+(-\mathbf{f}, \mathcal{U}, \mathbb{R}^n \setminus \mathbb{T})$  and  $\text{Bwd}_{\mathbf{f},\mathcal{U}}(\mathbb{T}) = \mathbb{R}^n \setminus \text{Viab}^+(\mathbf{f}, \mathcal{U}, \mathbb{R}^n \setminus \mathbb{T})$  which correspond to the maximal forward or maximal backward reach set.

**Example 4.10** (Car on the Hill). Another interesting system to consider is the case of the car on a hill system described by the following state equation (Lhommeau, Jaulin, and Hardouin, 2011) with an input:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -9.81 \sin(g(x_1)) - \alpha x_2 + u \end{cases} \quad (4.1)$$

where  $u$  is the input,  $\alpha$  a damping coefficient that will be set to 0.7 and  $g$  the shape of the hill which will be set to  $g(x) = \frac{1}{2}(1.1 \sin(1.2x) - 1.2 \sin(1.1s))$ . Figure 4.11 shows the shape of the hill.

We set the input to  $u = 2$ ,  $\mathbb{T} = \{\mathbf{x} \mid x_1^2 + x_2^2 \leq 1\}$  and  $[\mathbb{P}] = [-2, 18] \times [-4, 5]$ . The bracket of  $\text{Fwd}_{\mathbf{f}}(\mathbb{T})$  is shown in Figure 4.12. The result can be physically interpreted the following way: a car starts from the left part of the hill with its engine set to produce a force equal to  $u$ . The car is then

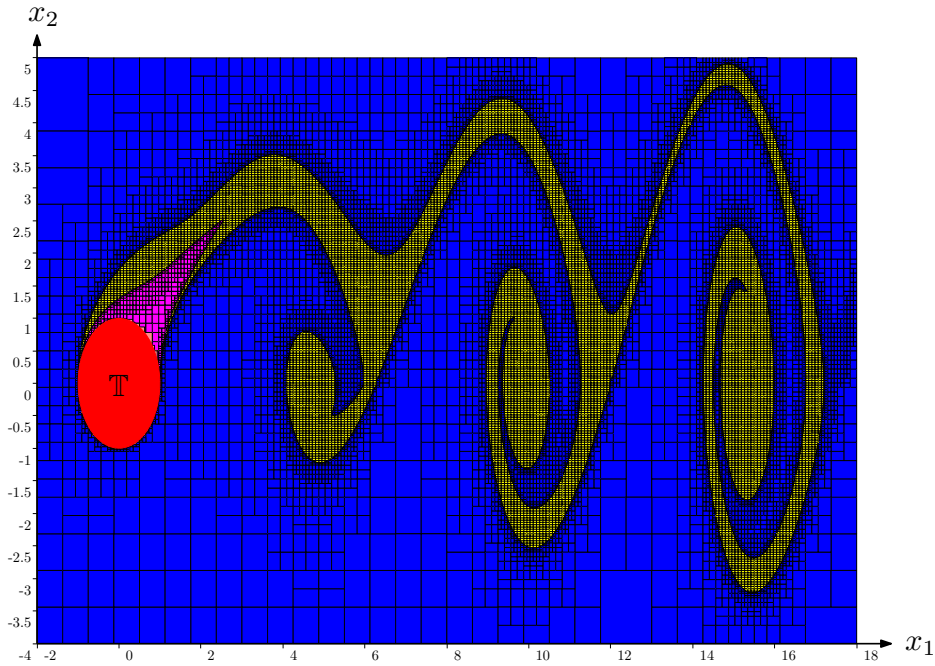


Figure 4.12: Car on the hill  $\text{Fwd}_f(\mathbb{T})$  set.

released inside a set of initial positions and velocities. The result shows the set of position and velocities that can be reached by the system.

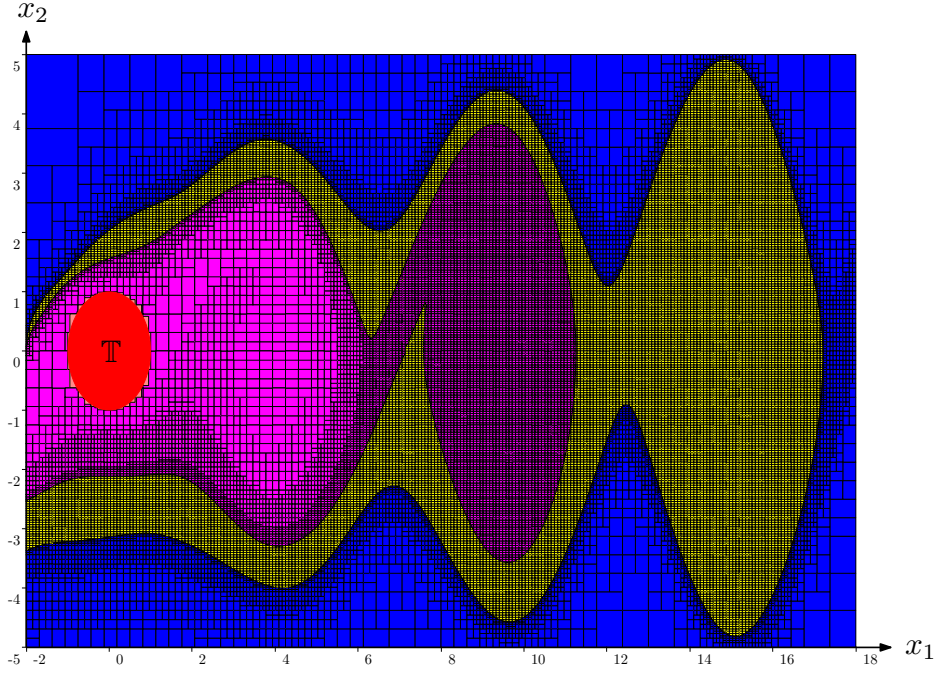
We can note that there are two stable equilibrium points around  $x_1 \simeq 4.5$  and  $x_1 \simeq 10$ , with  $x_2 = 0$ , which correspond to two valleys with the input  $u = 2$ . We can also note that there are two unstable equilibrium points around  $x_1 \simeq 6.2$  and  $x_1 \simeq 12$  also with  $x_2 = 0$  that correspond to two crests. The position of the stable and unstable points would be different for another input  $u$ .

If we now compute the maximal forward reach set  $\text{Fwd}_{f,\mathcal{U}}(\mathbb{T})$  with an input  $u \in \mathcal{U} = [-2, 2]$ , we obtain Figure 4.13. We can see that a larger part of the state space can be reached compared to the case where  $u$  was set to a constant. Note that in this case, the outer approximation is not guaranteed as it intersects the boundaries of  $\mathbb{P}$  at the left part of Figure 4.13 (see Remark 4.8).

**Example 4.11** (3D system). We consider a Dubins car (Dubins, 1957) described by the following state equations:

$$\begin{cases} \dot{x}_1 = -0.1 \cos(x_3) \\ \dot{x}_2 = -0.1 \sin(x_3) \\ \dot{x}_3 = -0.3 \end{cases} \quad (4.2)$$



Figure 4.13: Car on the hill  $\text{Fwd}_{\mathbf{f}, \mathcal{U}}(\mathbb{T})$  set.

with the target set  $\mathbb{T} = [-0.5, 0.5] \times [-0.5, 0.5] \times [0, 0.5]$ . Figure 4.14 shows the result with  $[\mathbb{P}] = [-10, 10] \times [-10, 10] \times [0, 2\pi]$ .

*Remark 4.12.* A comparison with existing methods should be undertaken in future works. We have compared for instance the *Car on the Hill* example with CAPD. However, this problem better works with Eulerian methods than with Lagrangian methods as several different paths can be reached from the initial state. Therefore, we obtain a better approximation of the set.

### 4.3 Attraction basin

For some systems, we may want to know the set of all paths that will reach and then stay inside an area forever. The *backward reach set*,  $\text{Bwd}_{\mathbf{f}}(\mathbb{T})$ , presented previously is here not sufficient to solve the problem. Indeed, paths can leave the set  $\mathbb{T}$  after reaching it. We only have the guarantee with  $\text{Bwd}_{\mathbf{f}}(\mathbb{T})$  that the intersection between the path and  $\mathbb{T}$  is not empty. This is why we introduce the *attraction basin*.

**Definition 4.13.** The *attraction basin*,  $\mathcal{B}(\mathbb{T})$ , of a set  $\mathbb{T}$  is the set:

$$\mathcal{B}(\mathbb{T}) = \{\mathbf{x} \mid \exists t_1 \geq 0, \forall t \geq t_1, \varphi_{\mathbf{f}}(t, \mathbf{x}) \in \mathbb{T}\}. \quad (4.3)$$

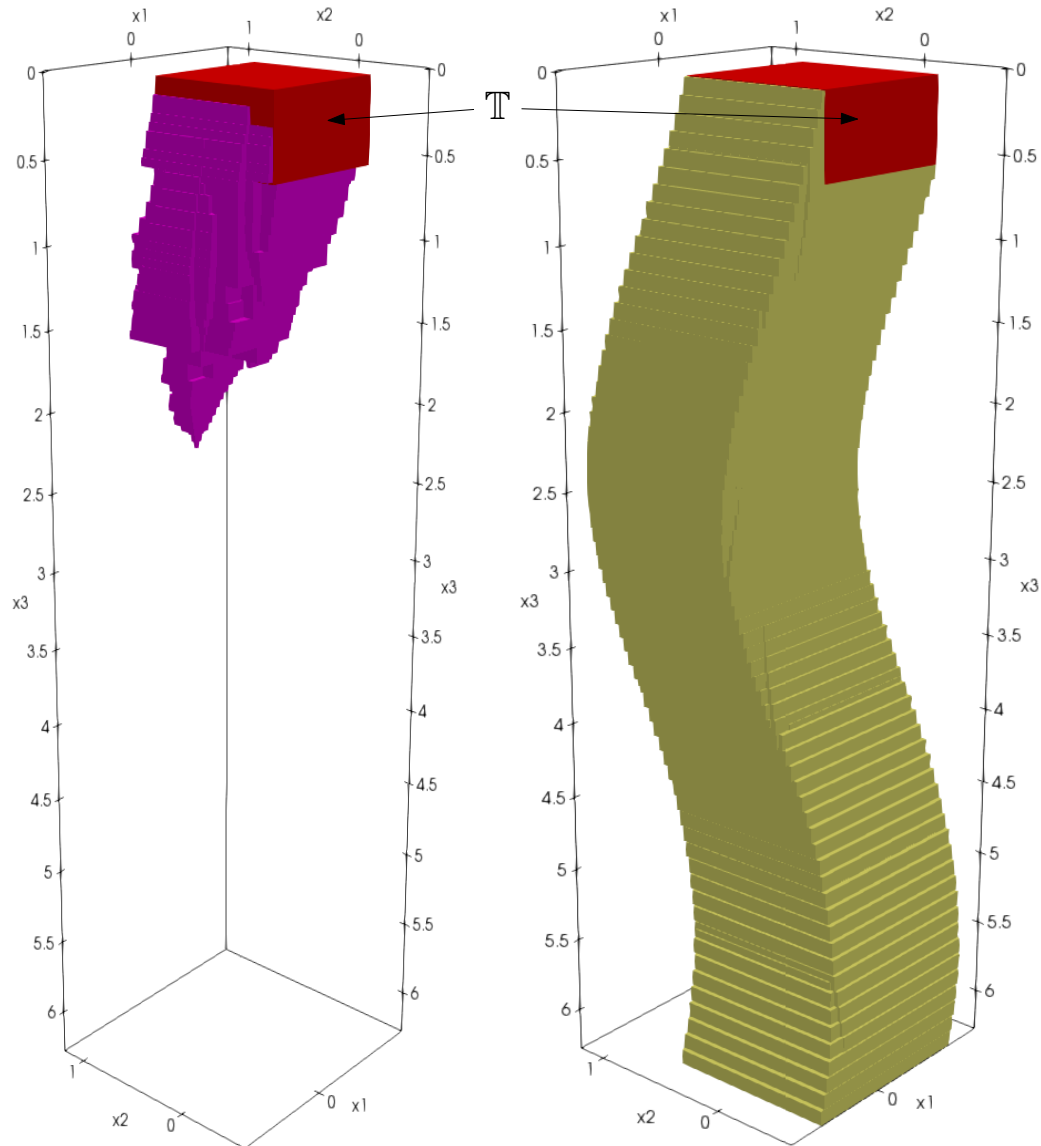


Figure 4.14: Bracketing the backward reach set of  $\mathbb{T}$  (the red box at the top) for a Dubins system. On the left, the inner approximation is magenta painted. On the right, the outer approximation is yellow painted.

With this definition, when the system reaches  $\mathbb{T}$ , it will stay forever inside  $\mathbb{T}$ . We recall that the *attraction basin* is different from the backward reach set as in the case of the latter it is only required to reach  $\mathbb{T}$  at one time but there is not the constraint to stay inside  $\mathbb{T}$  (see Definition 4.4).

**Theorem 4.14.** *We have:*

$$\mathcal{B}(\mathbb{T}) = \text{Bwd}\left(\overline{\text{Bwd}(\mathbb{T})}\right) \quad (4.4)$$

*Proof.* Let  $\mathbf{x}$  be a point of  $\mathcal{B}(\mathbb{T})$ , we have:

$$\begin{aligned} \mathbf{x} \in \mathcal{B}(\mathbb{T}) &\Leftrightarrow \exists t_1 \geq 0, \forall t \geq t_1, \varphi_{\mathbf{f}}(t, \mathbf{x}) \in \mathbb{T} \\ &\Leftrightarrow \exists t_1 \geq 0, \neg(\exists t \geq t_1, \varphi_{\mathbf{f}}(t, \mathbf{x}) \in \overline{\mathbb{T}}) \\ &\Leftrightarrow \exists t_1 \geq 0, \varphi_{\mathbf{f}}(t_1, \mathbf{x}) \in \overline{\text{Bwd}(\mathbb{T})} \\ &\Leftrightarrow \mathbf{x} \in \text{Bwd}\left(\overline{\text{Bwd}(\mathbb{T})}\right) \end{aligned}$$

□

**Example 4.15.** Let us consider a buckling system which models the instability of an axially compressed mechanical structure. The system can be described by the following equations:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -0.1(20x_1^3 - 10x_1 + 5x_2) \end{cases} \quad (4.5)$$

and a target set  $\mathbb{T} = \{\mathbf{x} \mid (x_1 + 0.5)^2 + (x_2)^2 \leq 1\}$ . The system has two stable equilibrium points in  $\mathbf{x} \approx (-0.707, 0)^\top$  and  $\mathbf{x} \approx (0.707, 0)^\top$ , and an unstable equilibrium point in  $\mathbf{x} = (0, 0)^\top$ . Figure 4.15 shows the vector field of the system where the target set is the red disk.

Figure 4.16 gives a bracket of  $\text{Bwd}(\mathbb{T})$ . Figure 4.17a provides a bracket of  $\overline{\text{Bwd}(\mathbb{T})}$  which also corresponds to the largest positive invariant set included in  $\mathbb{T}$ . Figure 4.17b shows the attraction basin  $\text{Bwd}\left(\overline{\text{Bwd}(\mathbb{T})}\right)$ . We clearly see the difference between the backward reach set and the attraction basin. Note that we have been using a shared subpaving for all the mazes which explains the bisection for  $\overline{\text{Bwd}(\mathbb{T})}$ .

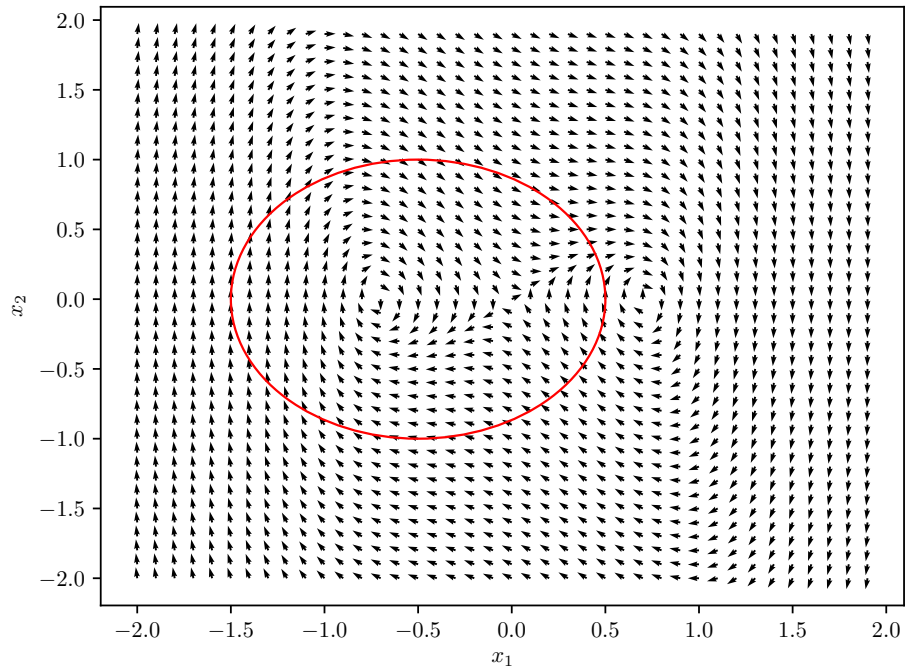
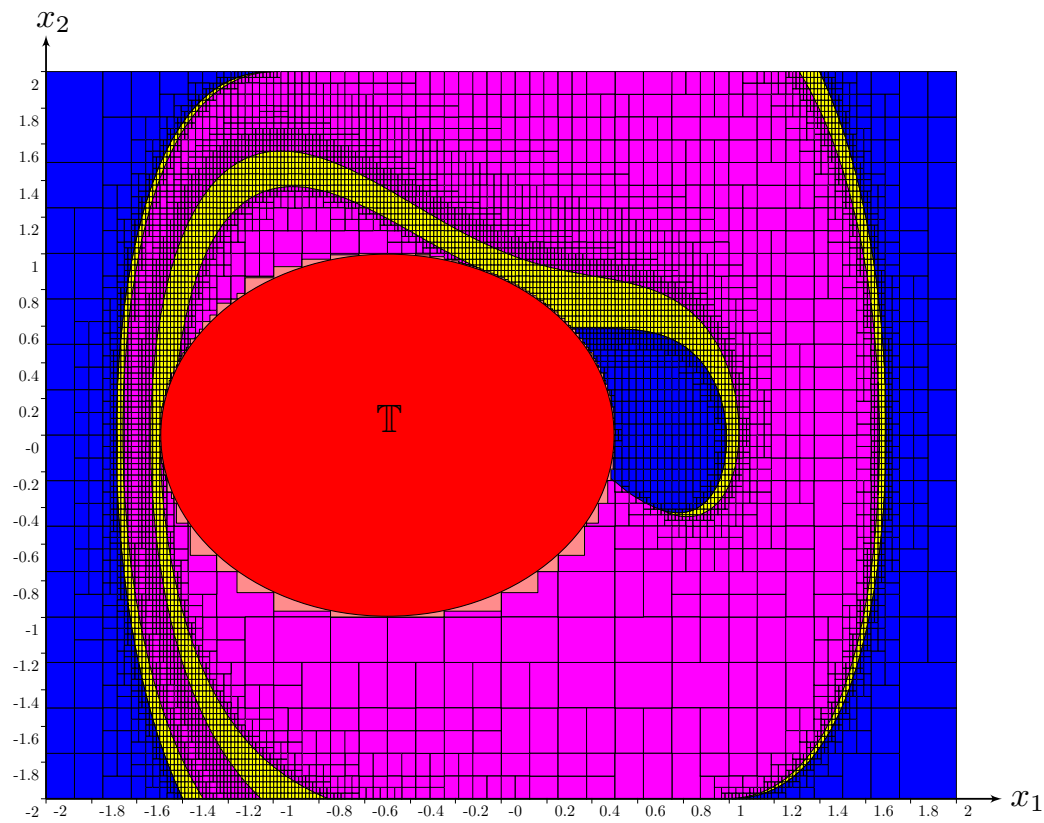


Figure 4.15: Vector field of the buckling system.

Figure 4.16: Backward reach set  $Bwd(\mathbb{T})$ .

## 4.4 Capture reach set

The fourth problem we will consider, is the bracket of a *capture reach set*. Solving this problem will allow us to consider constraints of the form: “the path avoid a set of the state space” in the next section.

**Definition 4.16.** The *capture forward reach set*,  $Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T})$ , of a set  $\mathbb{T}$  is the set:

$$Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T}) = \left\{ \mathbf{x} \mid \exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, (\mathbf{x} = \varphi_{\mathbf{f}}(t, \mathbf{x}_0)) \right. \\ \left. \wedge (\forall s \in [0, t], \varphi_{\mathbf{f}}(s, \mathbf{x}_0) \in \mathbb{C}) \right\},$$

where  $\mathbb{T}, \mathbb{C} \subseteq \mathbb{R}^n$ .

**Example 4.17.** Figure 4.18 illustrates a capture forward reach set.  $\mathbb{C}$  is the whole space except the hatched area. We can see that paths are stopped when they intersect the boundary of  $\mathbb{C}$ . Note that there is no need to have  $\mathbb{T} \subseteq \mathbb{C}$ . Indeed, we have:  $Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T}) = Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T} \cap \mathbb{C})$  and  $\mathbb{T} \cap \mathbb{C} \subseteq \mathbb{C}$ .

**Theorem 4.18.** *We have:*

$$Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T}) = Fwd_{\mathbf{g}}(\mathbb{T} \cap \mathbb{C}) \quad (4.6)$$

$$\text{where } \mathbf{g}(\mathbf{x}) = \begin{cases} \mathbf{f}(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbb{C} \\ \mathbf{0} & \text{otherwise} \end{cases}.$$

*Proof.* Let  $\mathbf{x}$  be a point of  $Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T})$ , we have:

$$\begin{aligned} \mathbf{x} \in Capt_{\mathbb{C},\mathbf{f}}^+(\mathbb{T}) &\Leftrightarrow \exists \mathbf{x}_0 \in \mathbb{T}, \exists t \geq 0, \mathbf{x} = \varphi_{\mathbf{f}}(t, \mathbf{x}_0) \wedge \mathbf{x} \in \mathbb{C} \\ &\Leftrightarrow \exists \mathbf{x}_0 \in \mathbb{T} \cap \mathbb{C}, \exists t \geq 0, \mathbf{x} = \varphi_{\mathbf{g}}(t, \mathbf{x}_0) \\ &\Leftrightarrow Fwd_{\mathbf{g}}(\mathbb{T} \cap \mathbb{C}) \end{aligned}$$

□

**Example 4.19.** To illustrate the capture reach set, we take back the Van Der Pol system with the same conditions as in Example 4.7. We set  $\mathbb{C} = \{\mathbf{x} \mid (x_1 - 1.6)^2 + (x_2)^2 \geq (0.2)^2\}$  which corresponds to the complement of a disk. The result is shown in Figure 4.19 where the green hatched disk is the complement of  $\mathbb{C}$ .

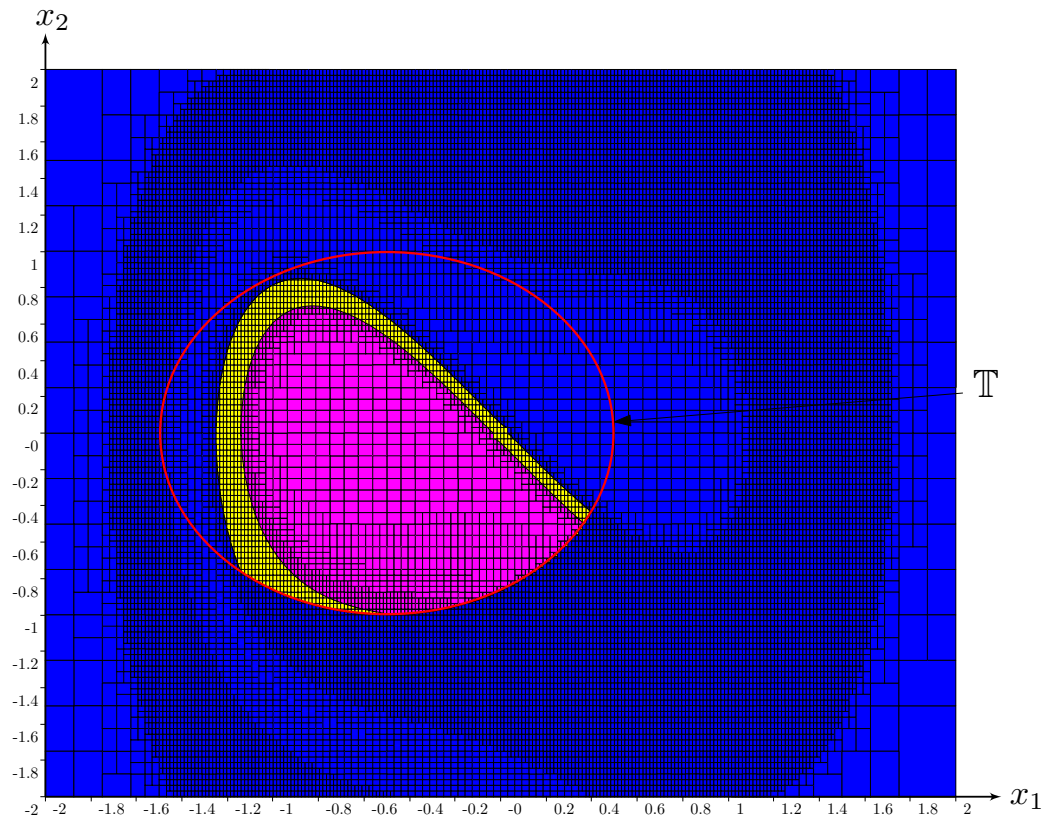
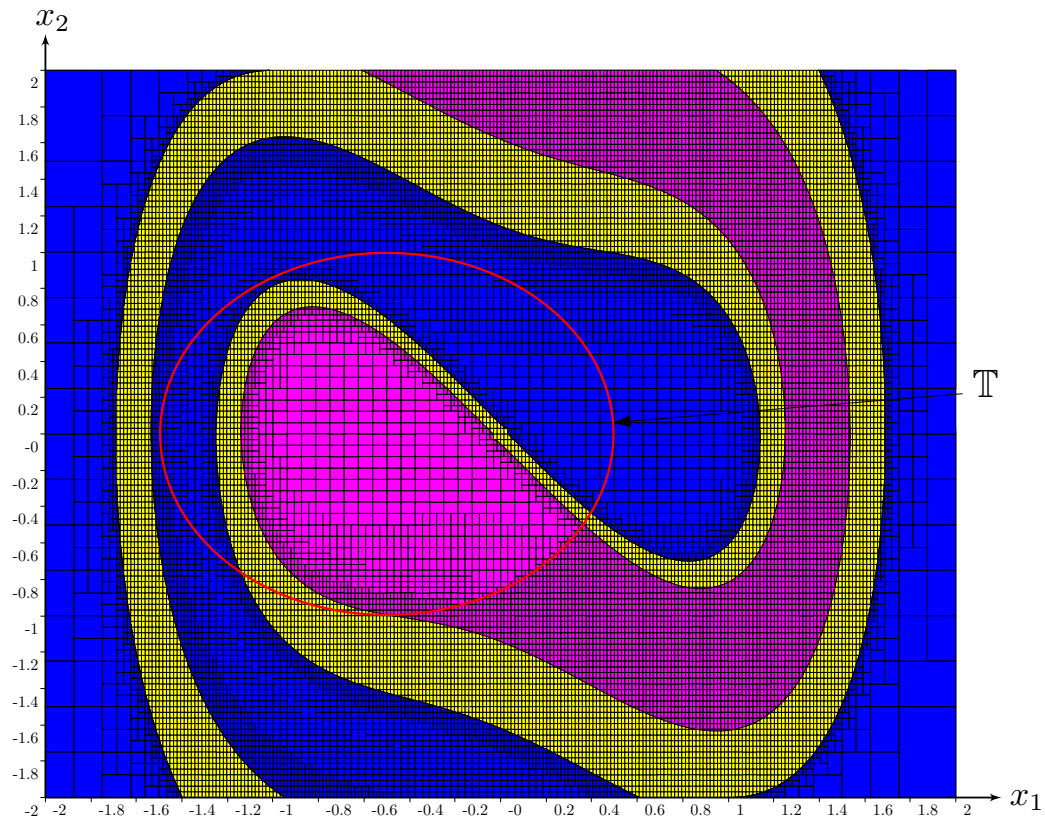
(a) The largest positive invariant set  $\overline{Bwd(\mathbb{T})}$  included in  $\mathbb{T}$ .(b) Bracket of  $\overline{Bwd(\mathbb{T})}$ .

Figure 4.17: Attraction basin of the buckling system.

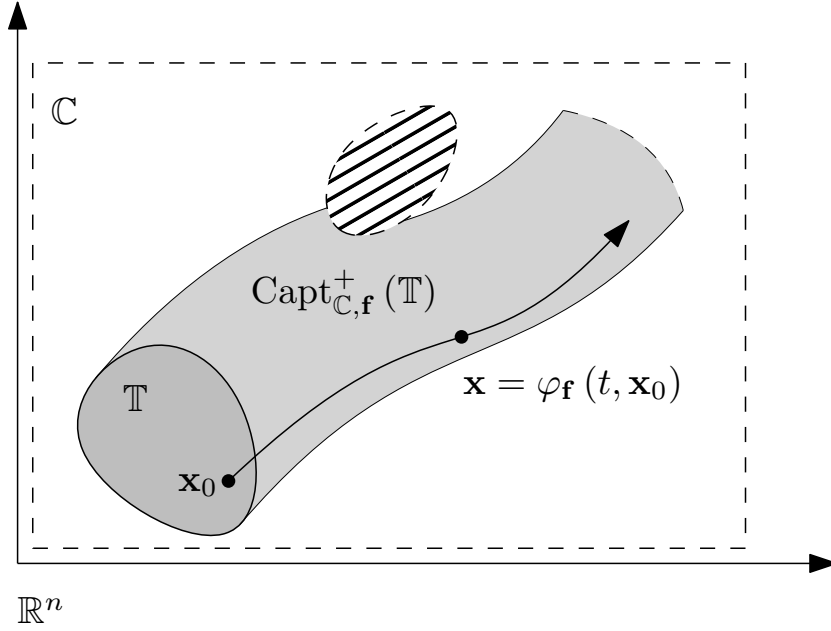


Figure 4.18: Illustration of the capture reach set.

*Remark 4.20.* In our implementation, the  $\mathbf{0}$  value of the vector field is implemented as a special value and not as a numerical value. By this way, the algorithm will not inflate doors with  $\text{FLOWINFLATEBWD}_{\mathbf{f}}(\mathcal{L})$  when  $\mathbf{0}$  is the only value of the box vector field. It will similarly close all doors of a box when using  $\text{FLOWCONTRACTBWD}_{\mathbf{f}}(\mathcal{L})$ .

We can also define the *capture backward reach set*  $\text{Capt}_{\mathbf{C},\mathbf{f}}^-(\mathbb{T})$ .

## 4.5 Eulerian state estimation

In order to do a synthesis of the tools presented in this chapter, we will introduce here the *Eulerian state estimation* problem (Le Mézo, Jaulin, and Zerr, 2017b) which formalizes all the previous problems in a unique framework.

In the context of an Eulerian point of view, we would like to be able to deal with constraints such as:

- *Robot A* has met *robot B* before the collision with *robot C*,
- After the robot turn in the vicinity of the buoy, its speed was always lower than  $1 \text{ m s}^{-1}$ .

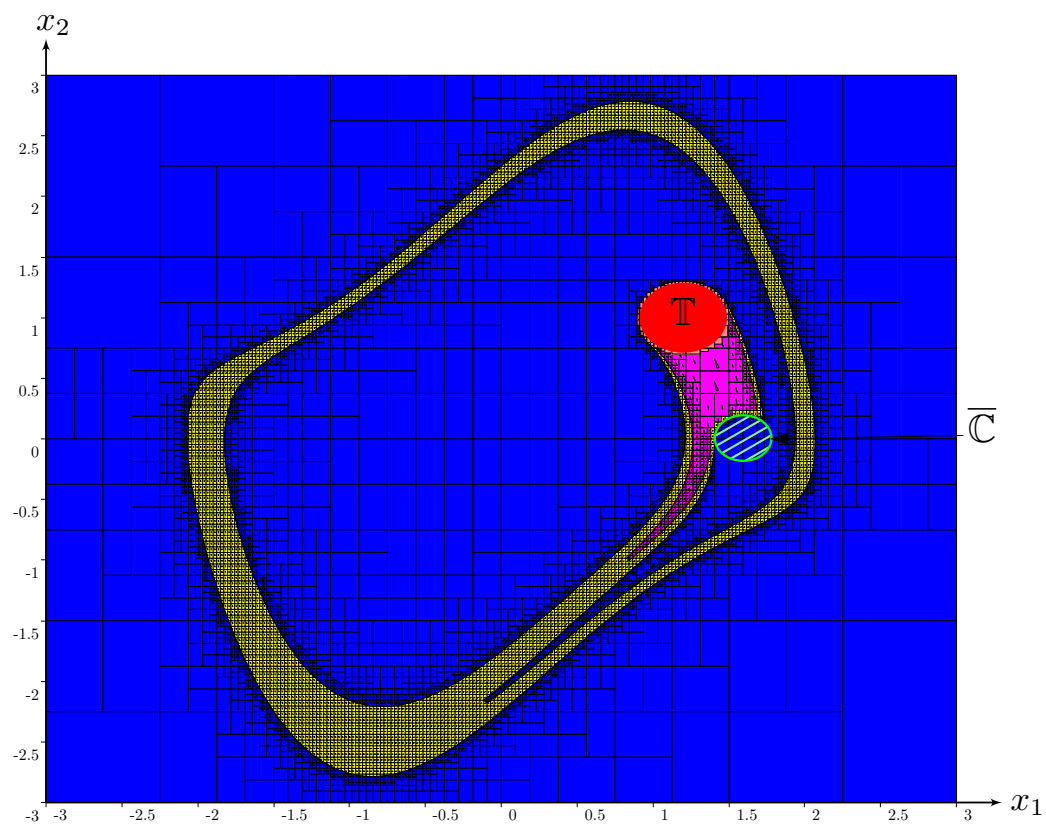


Figure 4.19: Capture reach set for the Van Der Pol System.



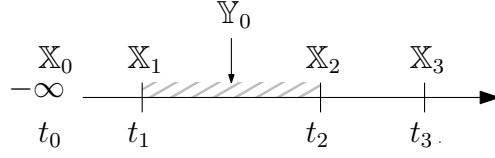


Figure 4.20: Example of constraints for the *Eulerian state estimation* problem.

### 4.5.1 Formalism

We model the *Eulerian state estimation* problem with the following CN where the variables are paths of  $\mathbb{R}^n$ , the domain is a maze and the constraints are:

$$\left\{ \begin{array}{ll} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) & \text{(evolution)} \\ \mathcal{E} = (t_0, \dots, t_l), t_i \leq t_{i+1} & \text{(precedence)} \\ \mathbf{x}(t_i) \in \mathbb{X}_i \subset \mathbb{R}^n & \text{(event)} \\ \mathcal{C} \subset \mathcal{E}, \forall t_k \in \mathcal{C}, \forall t \in [t_k, t_{k+1}], \mathbf{x}(t) \in \mathbb{Y}_k & \text{(capture)} \end{array} \right. ,$$

where  $\mathbb{X}_i$  and  $\mathbb{Y}_i$  are sets of  $\mathbb{R}^n$ ,  $\mathcal{E}$  is an ascending list of  $l$  times with  $t_i \in [-\infty, +\infty]$  and  $\mathcal{C}$  is a subset of  $\mathcal{E}$ . In this problem we do not know the  $t_i$  but only their order as we adopted an Eulerian point of view.

**Example 4.21.** Figure 4.20 shows an example of an *Eulerian state estimation* problem with a sequence of 4 times:  $\mathcal{E} = (t_0, t_1, t_2, t_3)$  and  $\mathcal{C} = (t_1)$ . Note that here  $t_0 = -\infty$ . Figure 4.21 shows several paths in the state space along with the sets  $\mathbb{X}_i$  and  $\mathbb{Y}_i$ . (i) starts from a limit cycle in  $\mathbb{X}_0$ , crosses  $\mathbb{X}_1$  then  $\mathbb{X}_2$  while staying in  $\mathbb{Y}_0$  between the events  $t_1$  and  $t_2$ , and finally ends in  $\mathbb{X}_3$  with an equilibrium point. (ii) also verifies the constraints even if it crosses  $\mathbb{X}_2$ , between  $\mathbb{X}_0$  and  $\mathbb{X}_1$ , which is not forbidden. (iii) does not verify the constraints as it does not start inside  $\mathbb{X}_0$  and does not cross  $\mathbb{X}_2$ .

### 4.5.2 Invariant sets approach

We define the set  $\mathbb{Z}_i^{\text{fwd}}$  of all state vectors  $\mathbf{x}(t)$  inside  $\mathbb{X}_i$  that have visited  $\mathbb{X}_0, \mathbb{X}_1, \dots, \mathbb{X}_{i-1}$  in the past, *i.e.* before time  $t$ , in this specific order and that verify the capture conditions. We define the following sequence as the *Eulerian filter*:

$$\left\{ \begin{array}{l} \mathbb{Z}_{i+1}^{\text{fwd}} = \begin{cases} \text{Capt}_{\mathbb{Y}_i, \mathbf{f}}^+(\mathbb{Z}_i^{\text{fwd}}) \cap \mathbb{X}_{i+1} & \text{if } i \in \mathcal{C} \\ \text{Fwd}_{\mathbf{f}}(\mathbb{Z}_i^{\text{fwd}}) \cap \mathbb{X}_{i+1} & \text{else} \end{cases} \\ \mathbb{Z}_0^{\text{fwd}} = \begin{cases} \text{Inv}_{\mathbf{f}}^-(\mathbb{X}_0) & \text{if } t_0 = -\infty \\ \mathbb{X}_0 & \text{else} \end{cases} \end{array} \right. .$$

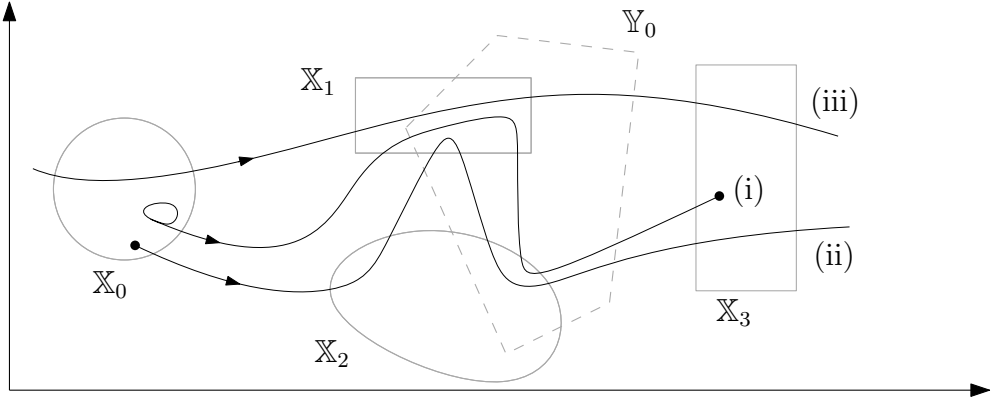


Figure 4.21: Paths of  $\mathbb{R}^n$  with the *Eulerian state estimation* problem.

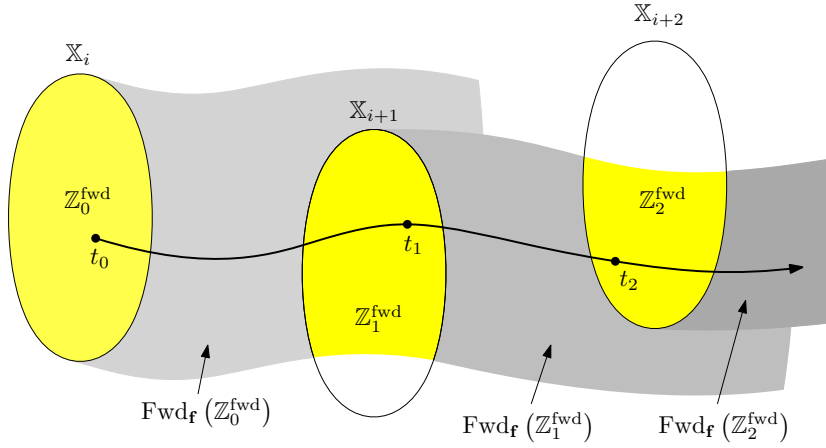


Figure 4.22: Illustration of the *Eulerian filter*.

Figure 4.22 illustrates the *Eulerian filter* in the case where we do not have any capture conditions and with  $t_0 \neq -\infty$ . The  $Z_i^{\text{fwd}}$  are represented in yellow. We can see that if we take a state  $\mathbf{x}_2 \in Z_2^{\text{fwd}}$ , there exist  $\mathbf{x}_1, \mathbf{x}_0$  such that  $\exists (t_2, t_1, t_0), (\mathbf{x}_1 = \varphi_{\mathbf{f}}(t_1, \mathbf{x}_0)) \wedge (\mathbf{x}_2 = \varphi_{\mathbf{f}}(t_2, \mathbf{x}_0)) \wedge (t_0 \leq t_1 \leq t_2)$ .

In the case where we have a capture condition, we replace the  $\text{Fwd}_{\mathbf{f}}(Z_i^{\text{fwd}})$  set by a  $\text{Capt}_{\mathbb{Y}_i, \mathbf{f}}^+(Z_i^{\text{fwd}})$  set as presented in Section 4.4. In the case where  $t_0 = -\infty$ , this means that the path has stayed for all negative times in the set  $X_0$  which is equivalent to the largest negative invariant set.

Similarly, we define the set  $Z_i^{\text{bwd}}$  of all state vectors  $\mathbf{x}(t)$  inside  $X_i$  that have visited  $X_l, X_{l-1}, \dots, X_{i+1}$ . We define the following sequence as the *Eulerian smoother*:

$$\begin{cases} \mathbb{Z}_{i-1}^{\text{bwd}} = \begin{cases} \text{Capt}_{\mathbb{Y}_{i,\mathbf{f}}}^-(\mathbb{Z}_i^{\text{bwd}}) \cap \mathbb{X}_{i-1} & \text{if } i \in \mathcal{C} \\ \text{Bwd}_{\mathbf{f}}(\mathbb{Z}_i^{\text{bwd}}) \cap \mathbb{X}_{i-1} & \text{else} \end{cases} \\ \mathbb{Z}_l^{\text{bwd}} = \begin{cases} \text{Inv}_{\mathbf{f}}^+(\mathbb{X}_l) & \text{if } t_l = +\infty \\ \mathbb{X}_l & \text{else} \end{cases} \end{cases} .$$

The solution set of the *Eulerian state estimation* problem is the set:

$$\mathbb{S} = \text{Fwd}_{\mathbf{f}}(\mathbb{Z}_0^{\text{bwd}}) \cap \text{Bwd}_{\mathbf{f}}(\mathbb{Z}_l^{\text{fwd}}) .$$

Figure 4.23 illustrates the previous formula.

*Remark 4.22.* We could have also taken  $\mathbb{Z}_l^{\text{bwd}} = \mathbb{Z}_l^{\text{fwd}}$  to initialize the *Eulerian smoother*. The *Eulerian state estimator* could then have been computed with  $\mathbb{S} = \text{Fwd}_{\mathbf{f}}(\mathbb{Z}_0^{\text{bwd}}) \cap \text{Bwd}_{\mathbf{f}}(\mathbb{Z}_l^{\text{bwd}})$ . We have been using this approach in the implementation. Each set  $\mathbb{Z}$  is represented by an instantiation of the maze which all rely on the same subpaving. To reach a fixed point, we must guarantee that constraints are propagated to all sets: this is why a forward propagation and then a backward propagation was preferred rather than the intersection of the *smoother* and *filter*. Figure 4.24 shows how the propagation was implemented.

**Example 4.23.** Let us consider again the Van Der Pol system and two events  $\mathcal{E} = (t_0, t_1)$  with the associated sets:  $\mathbb{X}_0 = \{\mathbf{x} \mid (x_1 - 0.8)^2 + (x_2 - 1.3)^2 \leq (0.4)^2\}$ ,  $\mathbb{X}_1 = [0.74, 1.2] \times [-1.5, -1.06]$ . We set the capture constraint to  $\mathcal{C} = \{t_0\}$  with  $\mathbb{Y}_0 = [1.35, 1.45] \times [-0.2, 0]$ . We search for paths that validate all the constraints. This problem can be rewritten in terms of invariants by bracketing the set:

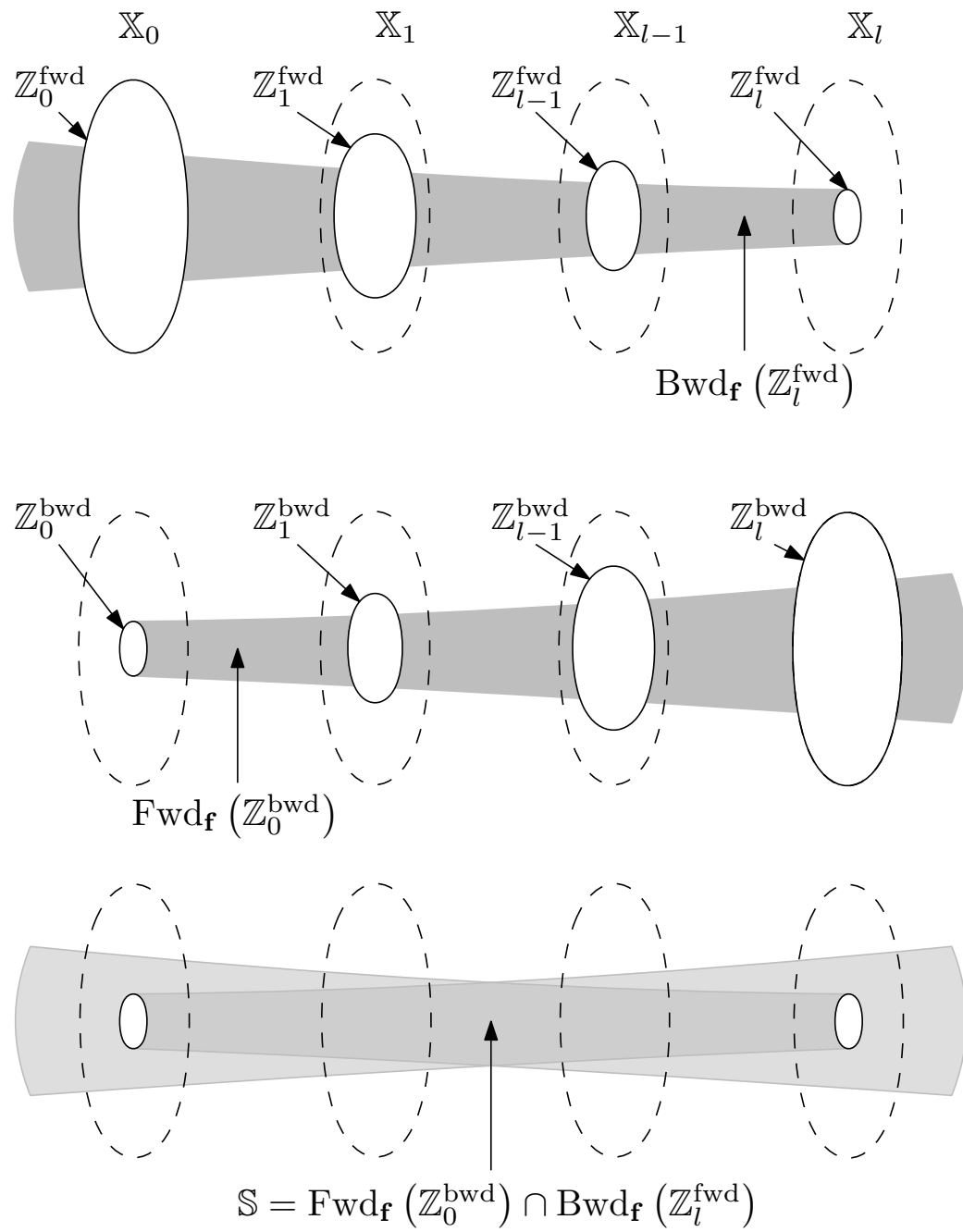
$$\mathbb{S} = \text{Fwd}_{\mathbf{f}}(\text{Capt}_{\mathbb{Y}_0, \mathbf{f}}^-(\mathbb{X}_1) \cap \mathbb{X}_0) \cap \text{Bwd}_{\mathbf{f}}(\text{Capt}_{\mathbb{Y}_0, \mathbf{f}}^+(\mathbb{X}_0) \cap \mathbb{X}_1) .$$

Figure 4.25 shows the result with the outer approximation in yellow and the inner approximation in magenta.

*Remark 4.24.* We could have added logical disjunction to the constraints such as: “the paths should visit  $\mathbb{X}_1$  OR  $\mathbb{X}_2$ ” which can be solved by computing the union of sets.

**Example 4.25** (Eulerian state estimator and ocean currents). Let us consider a real dataset<sup>8</sup> of ocean currents near Ouessant island that we will

<sup>8</sup>MARC\_L1-MARS2D-FINIS250 model with  $t_0 = 20170709T0000$

Figure 4.23: Illustration of the *Eulerian state estimator*.

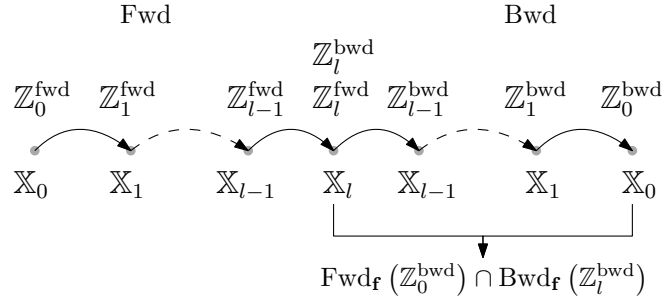


Figure 4.24: Propagation of constraints in the *Eulerian state estimator*.

suppose accurate.<sup>9</sup> Let us assume that a profiling float<sup>10</sup> has sent an uncertain position at three uncertain times. We want to find the set of all possible paths during the mission.

The state space here is  $(t, x, y)$  for the time (in s), the east coordinate (in m) and north coordinate (in m). Indeed, the value of the currents is time-dependent. In this theoretical example, we assume that the three observation sets are distributed along 10.5 hours:  $X_0 = [0, 300] \times [33250, 35250] \times [118750, 119750]$ ,  $X_1 = [18000, 28000] \times [43100, 46100] \times [122000, 125000]$  and  $X_2 = [37500, 37800] \times [32250, 34250] \times [117250, 119250]$ . Figure 4.26 shows the outer approximation of the solution set. The dataset contractor (see Section 2.3.3.2 on page 71) was used here to evaluate efficiently the evolution function built on discrete data. The computation time is around 5 hours on a 24 cores CPU. Polytopes have been chosen to be the abstract domain for doors.

*Remark 4.26.* We can draw a link between Eulerian filter problems and the temporal logic field (Huth and Ryan, 2004) such as the LTL (linear temporal logic). For instance, a problem that verifies that there exist paths starting from a set<sup>11</sup>  $s$  that reach a set  $p$ , is expressed as  $\mathbf{EF}p$  in LTL and can be formulated as finding if the set solution of the Eulerian filter problem with  $X_0 = s$ ,  $X_1 = p$ , is not empty. However, the temporal logic language is suited to work with discrete time systems and is not well suited to continuous time systems.

*Remark 4.27.* The Eulerian filter is not an efficient method to find a candidate

<sup>9</sup>For this example, we do not have computed an accurate projection to a cartesian coordinate which would have been necessary in a real application. The origin of the coordinate system is taken at the zero of the model grid. We consider that the grid is cartesian with a distance of 250 meters spatially and 300 seconds temporally between each vertex.

<sup>10</sup>see the robot taxonomy in Figure 1.2 on page 7

<sup>11</sup>The notation is taken from the LTL field.

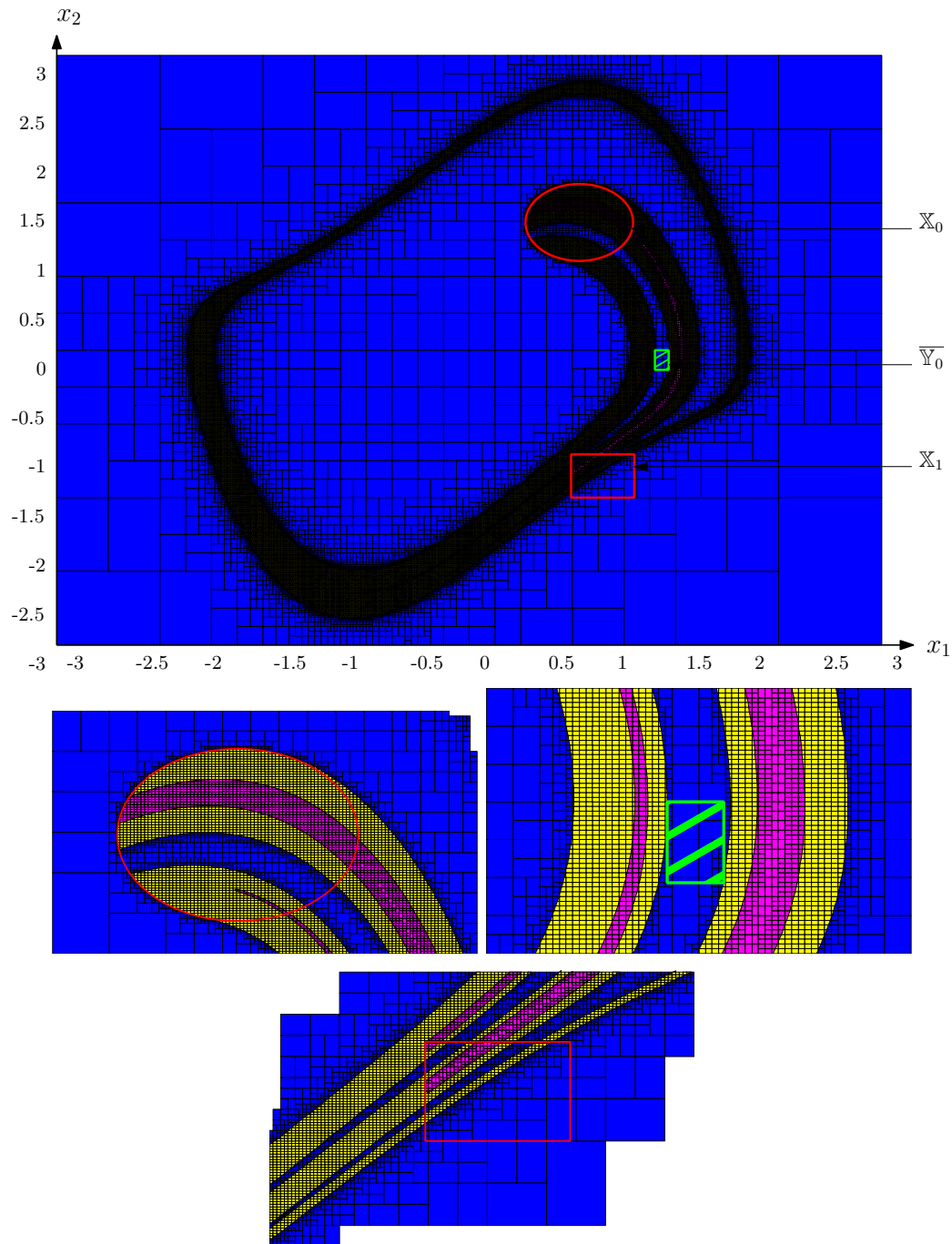


Figure 4.25: The Eulerian state estimator with the Van Der Pol system and zoom on  $X_0$ ,  $X_1$  and  $\bar{Y}_0$ .

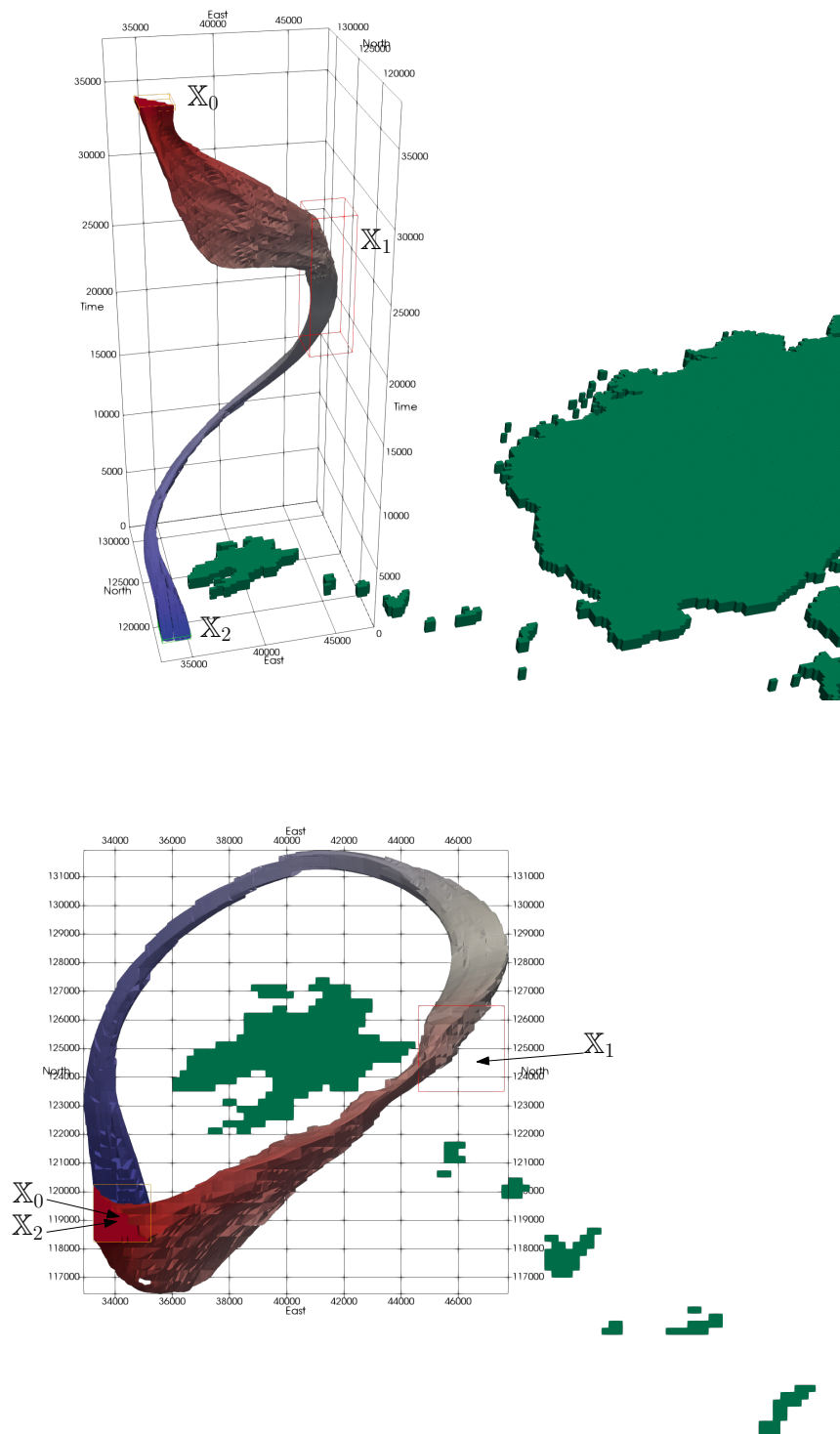


Figure 4.26: Example of the Eulerian state estimator with a dataset. The z-axis represents the time.

path for the *Challenge 1* presented in Section 1.1.2 on page 9. It can however be useful to prove that no paths exist, or at least to give a set in which candidate paths can be searched.

## 4.6 Conclusion

In this chapter, we have seen how positive invariant sets can be used to solve various problems. This was allowed by the lattice structure of mazes that enables to compute the intersection and the union of set of paths.

We have also seen through examples such as the isobath navigation problem or the problem of the Eulerian state estimator applied to ocean currents, how the tools presented in this chapter can solve robotic problems. They allow to prove the safety of systems which have or do not have inputs.

A perspective of this chapter would be to formalize an efficient language that could express paths constraints along with an interpreter which could convert the sentences of the language into a set of positive invariants sets to bracket.



# Chapter 5

## Design and control of a low-cost hybrid profiling float

### Contents

---

<b>5.1</b>	<b>Problem formalization . . . . .</b>	<b>158</b>
5.1.1	The mission . . . . .	158
5.1.2	The robot . . . . .	161
<b>5.2</b>	<b>Float dynamics . . . . .</b>	<b>162</b>
<b>5.3</b>	<b>Design of a hybrid profiling float . . . . .</b>	<b>165</b>
5.3.1	Mechanical architecture . . . . .	165
5.3.2	Electronic architecture . . . . .	170
5.3.3	Software architecture . . . . .	171
<b>5.4</b>	<b>Depth controller . . . . .</b>	<b>171</b>
5.4.1	Control law . . . . .	172
5.4.2	Estimation of unknown parameters . . . . .	174
5.4.3	Experimental results . . . . .	175
<b>5.5</b>	<b>Validation of the depth control law . . . . .</b>	<b>181</b>
5.5.1	Open-loop float performances . . . . .	181
5.5.2	Closed-loop system . . . . .	185
5.5.3	Additional validations . . . . .	187
<b>5.6</b>	<b>Design loop . . . . .</b>	<b>189</b>
<b>5.7</b>	<b>Conclusion and future work . . . . .</b>	<b>190</b>

---

This chapter mainly focuses on answering the third *Challenge* presented in Section 1.1.2 on page 9. We will consider the problem of designing a mission strategy and also a robot that enables to follow ocean currents. This chapter is less theoretical than the previous one. However, it is aimed at showing how *mazes* can help in practice to the design of robots and to the validation of their control laws.

## 5.1 Problem formalization

In this section we will mainly focus on formalizing the mission strategy and we will give the main features that the robot should fulfill. We did not reach the goal of implementing and validating the whole mission strategy due to a lack of time. We therefore present here only a global strategy and the results obtained so far.

### 5.1.1 The mission

We assume that the mission takes place in a shallow water environment. Moreover, we also assume that the robot is designed to be low-cost.<sup>1</sup>

#### 5.1.1.1 Use case

To answer the *Challenge*, we have chosen a specific strategy which is shown in Figure 5.1. The robot is only localized at surface with a GNSS (Global Navigation Satellite System). Indeed, underwater localization systems are expensive and should then be avoided. The robot dives to a constant depth where oceanic models are assumed to be valid. This implies not being near the surface nor near the seabed. This is why a depth between 10 and 20m was chosen as we are in a coastal environment. Finally, we will assume, to simplify the problem, that all paths corrections are carried out only at surface during a finite window time using auxiliary thrusters.

There are two operating modes for the robot: a waypoint following mode at surface<sup>2</sup>, and a stabilized depth mode when diving.

#### 5.1.1.2 Mission formalization

We propose the following global workflow for the mission (see Figure 5.2). We assume that we work in the state space  $\mathbb{R}^3$  with the north position ( $x_1$ ), the

---

<sup>1</sup>This is a key constraint for swarm of robots

<sup>2</sup>this mode will not be discussed in this work as it is a classic result of robotic systems

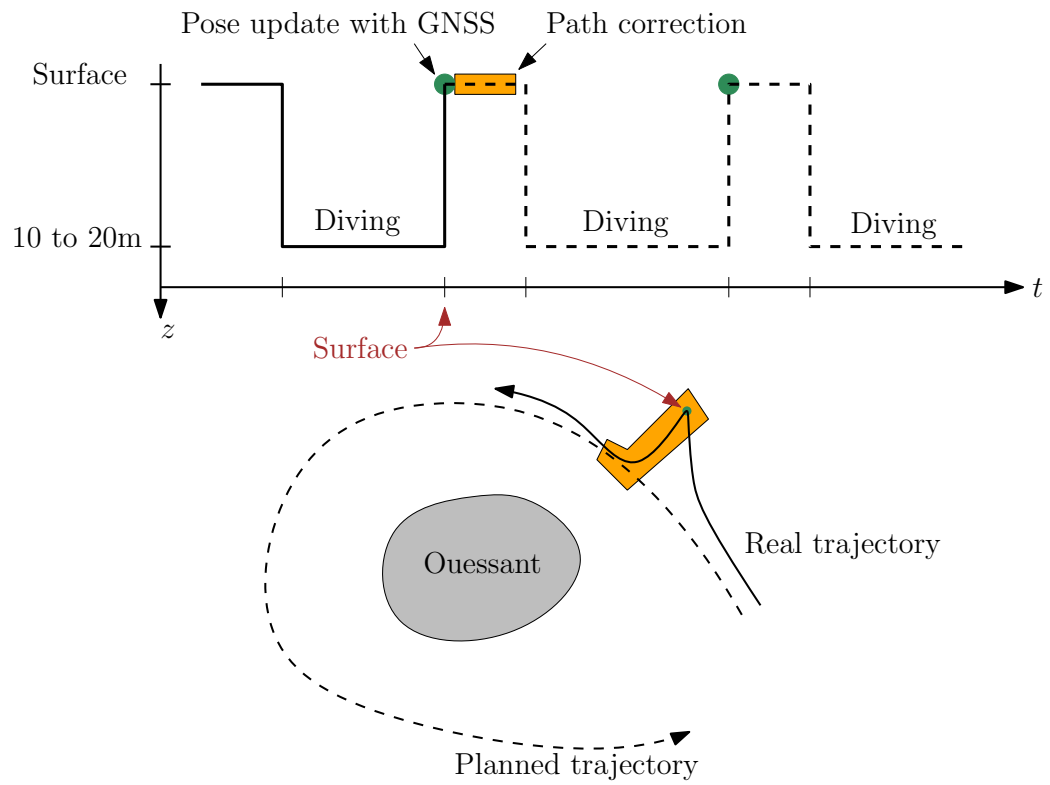


Figure 5.1: Strategy of the profiling float.

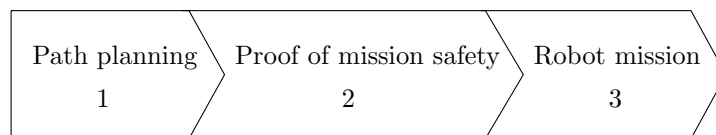


Figure 5.2: Mission planning workflow.

east position ( $x_2$ ) and the time ( $x_3$ ). We define a sequence of sets  $\{\mathbb{X}_i\}_{i \in [0, k]}$  through which the robot has to pass, and a set of sets  $\{\mathbb{Z}_0, \dots, \mathbb{Z}_l\}$  that the robot has to avoid. We also define a set of diving time intervals  $\mathbb{X}_D = \{[t_0], \dots, [t_k]\}$ .

Let us consider an oceanic current  $\dot{\mathbf{v}} = \mathbf{C}(\mathbf{x})$ , where  $\mathbf{x} \in \mathbb{R}^3$ , which outputs the velocity of the current in function of the state in a ground coordinate system. Let us also consider the global controller of the robot:

$$\dot{\mathbf{v}} = \mathbf{h}(\mathbf{x}) = \begin{cases} \mathbf{g}(\mathbf{x}) & \text{if } x_3 \in \mathbb{X}_D \\ \mathbf{0} & \text{else} \end{cases}$$

where  $\mathbf{g}$  is a simple waypoint controller that outputs the north and east velocity of the robot in function of the state in the water coordinate system<sup>3</sup>.

The robot path, in a ground coordinate system, is:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \mathbf{C}(\mathbf{x}) + \mathbf{g}(\mathbf{x}) \\ 1 \end{pmatrix}.$$

**Path planning** The path planning of AUV trajectories in ocean currents with uncertainties constraints has been the subject of several works (Rao and Williams, 2009; Smith et al., 2010; MahmoudZadeh et al., 2018). As stated in the introduction, due to a lack of time, we will not propose here any new method to solve the issue. We however propose a formalization of the constraints.

A candidate path  $\mathbf{p}$  of  $\mathbb{R}^3$  is generated along with a set of diving time intervals  $\mathbb{X}_D$ . The path should verify the following constraints<sup>4</sup>:

$$\begin{cases} \forall i \in [0, k], \exists s \in \mathbb{R}, & \mathbf{p}(s) \in \mathbb{X}_i \\ \forall s, \forall i \in [0, l], & \mathbf{p}(s) \cap \mathbb{Z}_i = \emptyset \end{cases} \quad (5.1)$$

**Example 5.1.** To illustrate that it may be possible to circumnavigate around Ouessant island, Figure 5.3 shows an example of paths in the case where there is no use of thrusters at surface. We can see that some paths achieve a circumnavigation. In this example, 125 paths were generated inside a box of 600 m wide spatially by 600 s wide temporally. We can see that the system is very sensitive to the initial state. Using thrusters should help to make the chosen path more robust to model and position uncertainties. The system is clearly chaotic.

<sup>3</sup>The thrusters modify the robot trajectory relatively to the surrounding water.

<sup>4</sup>Energy constraints could have also been added

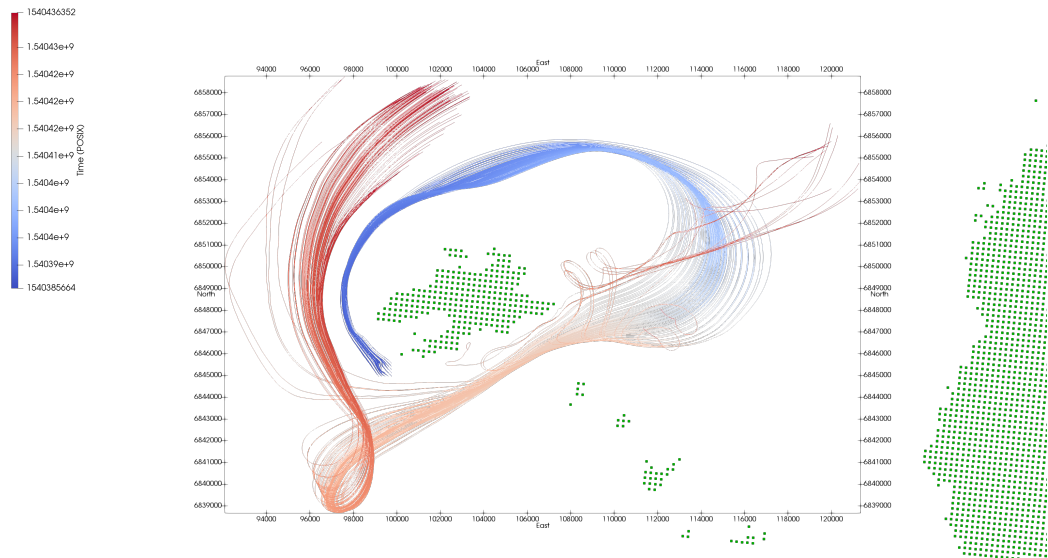


Figure 5.3: Monte Carlo simulation of paths, starting around  $t_0 = 1540386000$ ss (POSIX) and  $x_0 = 99474$  m,  $y_0 = 6845256$  m (Lambert93), MARC\_L1-MARS2D-FINIS250. Paths represent about 14 hours of mission.

**Path validation** The path is then validated to ensure that the mission can be fulfilled. This validation should take into account uncertainties of the model and uncertainties of the robot controller. This problem can be formalized in verifying that the *forward reach set* (see Section 4.2) of the system of dimension three:  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$  with an initial set  $\mathbb{T} = \mathbb{X}_0$  verifies the constraints of Equation 5.1.

A validation with a real case should be undertaken in future work. We did not have time to implement a complete example with a field validation, however the problem is very close to the Eulerian state estimator of Example 4.25 on page 151.

**Robot mission** The mission is then given to the robot. We must also ensure that low-level controllers, *i.e.* depth and waypoint controllers, are safe. The validation of the depth controller will be the subject of later Section 5.5.

### 5.1.2 The robot

As the main mode of operation is to be stabilized at constant depths, profiling floats (see Figure 1.2 on page 7) appear to be the most suitable kind of underwater robots to fulfill the mission.

**Profiling floats** Profiling floats can only regulate their depth and are widely used in oceanography. They are commonly equipped with instruments such as temperature, pressure, conductivity or biochemical sensors that measure the state of the water column. Most of them carry out profiles in the open ocean and the last generation can dive up to 4000 meters (Le Reste et al., 2016). They help to a better understanding of the ocean and provide crucial data to oceanographic models through several years missions. The most well-known profiling floats are those of the Argo project (Riser et al., 2016): about 4000 floats that gather data continuously all over the world (see Section 1.1.1 on page 6).

More recently, the oceanographic community has been focused on swarm of profiling floats for shallow water (Jaffe et al., 2017; Bessa et al., 2017) to better understand submesoscales dynamics ( $< 1 - 10\text{km}$ ). In shallow water, the vertical and horizontal variation of biochemical parameters can be important. This is why oceanographers seek to increase the density of data gathered.

**A hybrid profiling float** Current profiling floats cannot modify their horizontal path. We will propose in this chapter a new low-cost<sup>5</sup> hybrid profiling float called *SeaBot* that can, with auxiliary thrusters, correct horizontally its trajectory.

The remainder of this chapter will deal with the development of the robot and the validation of a low-level depth controller. After introducing the dynamic model of a float, a focus will be given on the design of the float. We will then look at the depth controller where a new control law, based on a full state feedback coupled with an Extended Kalman Filter (EKF), will be introduced and validated with experimental trials. An additional section will be dedicated to the application of *maze* tools to the system. We will finally propose a loop iteration design that are guidance rules to build such robots.

## 5.2 Float dynamics

A profiling float controls its buoyancy to regulate its vertical position. There exists several mechanical systems to perform this task that either adjust the mass or the volume of the float. They are mainly based on hydraulic pump or piston system. Some floats are also equipped with a passive system: they are designed to stabilize themselves at a unique density.

---

<sup>5</sup>By *low-cost* we mean a fast design and development phase, low or no calibration steps before using the float and obviously a low cost per unit. The cost of the whole life cycle should be taken into account.

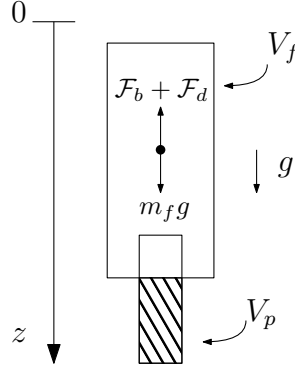


Figure 5.4: Float equipped with a piston system

We will choose the case of a piston based system (see Figure 5.4) which is simple to design and suitable for shallow water.

The principle is to adjust the volume of the float by pushing in or pulling out a piston that will modify the density and so the buoyancy. A float is primarily subject to gravity, buoyancy and drag forces. We make the assumption that the float has only vertical motion, with no rotation, that it is in thermal equilibrium with surrounding water, that the density of water is constant and that there is no vertical water velocity. A more complex model could be developed for more precise studies but a basic one seems to be sufficient to achieve an effective control (see the experimental results in Section 5.4.3 on page 175). We have:

$$(m_f + m_a) \ddot{z} = \mathcal{F}_b + \mathcal{F}_d - m_f g \quad (5.2)$$

where  $\mathcal{F}_b$  and  $\mathcal{F}_d$  are respectively the buoyancy force and the drag force.  $m_f$  is the mass of the float and  $m_a$  is the added mass which cannot be neglected in the case of water (Fossen, 2011). Table 5.1 summarizes all the parameters. The virtual mass  $m_v = m_f + m_a$  is the sum of the two masses. We then have

$$m_v \ddot{z} = -\rho g V_t - \frac{1}{2} C_d S \rho |\dot{z}| \dot{z} - m_f g \quad (5.3)$$

where  $V_t$  is the total volume of the float composed of the sum of the piston volume  $V_p$  and the float volume  $V_f$ . Note that the volume of the float  $V_f$  is supposed to be equal at zero depth to  $m_f/\rho$ : the float has a neutral buoyancy. The piston volume  $V_p$  is then defined as a positive or negative volume from neutral buoyancy at zero depth. Equation (5.3) can then be simplified to:

Parameter	Description	Unity	Typical <i>Seabot</i> value
$\ddot{z}$	float acceleration	$\text{m s}^{-2}$	
$\dot{z}$	float velocity	$\text{m s}^{-1}$	$[-0.3, 0.3]$
$z$	float depth	m	$[0, 50]$
$m_v$	virtual mass	kg	18
$m_f$	float mass	kg	9
$m_a$	added mass	kg	9
$\rho$	water density	$\text{kg m}^{-3}$	1025
$g$	acceleration due to gravity	$\text{m s}^{-2}$	9.81
$V_t$	total float volume	$\text{m}^3$	$V_f + V_p$
$V_f$	float volume	$\text{m}^3$	$\approx 8.8 \times 10^{-3}$
$V_p$	piston volume	$\text{m}^3$	$\Delta V_p = 1.7 \times 10^{-4}$
$S$	float's cross sectional area	$\text{m}^2$	$4.5 \times 10^{-2}$
$C_d$	drag coefficient	—	1
$\mathcal{K}_w$	water compressibility	$\text{Pa}^{-1}$	$4.27 \times 10^{-10}$
$\mathcal{K}_f$	float compressibility	$\text{Pa}^{-1}$	
$\chi$	loss of volume per meter	$\text{m}^2$	$2.15 \times 10^{-6}$

Table 5.1: Float physical parameters

$$\begin{aligned}
m_v \ddot{z} &= -\rho g (V_f + V_p) - \frac{1}{2} C_d S \rho |\dot{z}| \dot{z} - \rho V_f g \\
\ddot{z} &= -\frac{\rho g}{m_v} V_p - \frac{C_d S \rho}{2m_v} |\dot{z}| \dot{z}
\end{aligned} \tag{5.4}$$

A last phenomenon must be taken into account: the compressibility of the float. While increasing external pressure, the float's volume will decrease. This is also the case for water. The isothermal compressibility  $\mathcal{K}_T = -\frac{1}{V} \left( \frac{\partial V}{\partial P} \right)_T$  measures the relative change of volume as a response to a pressure. We will assume that the water and float temperature are constant which means that  $\mathcal{K}_w$ , the water compressibility, and  $\mathcal{K}_f$ , the float compressibility are constants. We will also assume that the relation between the pressure  $P$  and the depth  $z$  is linear equal to  $P(z) = \rho g z$ .

We can then deduce the loss of buoyancy of the float which is explained by the relative variation of volume  $\delta V$  of the float compared to the equivalent one of water under the same pressure:

$$\mathcal{F}_{\mathcal{K}} = \rho g \delta V = \rho g (\mathcal{K}_f - \mathcal{K}_w) P(z) V_f = (\mathcal{K}_f - \mathcal{K}_w) m_f \rho g^2 z.$$

Note that we have neglected the loss associated to the piston volume and we



have supposed that the volume of the float is constant. Equation (5.4) can then be rewritten to take into account the compressibility:

$$\ddot{z} = -\frac{\rho g}{m_v} V_p - \frac{C_d S \rho}{2m_v} |\dot{z}| z + (\mathcal{K}_f - \mathcal{K}_w) \frac{m_f \rho g^2 z}{m_v}.$$

We set  $\chi = m_f (\mathcal{K}_f - \mathcal{K}_w) g$ , the loss of volume per meter depth, which is homogeneous to  $\text{m}^2$ . We obtain

$$\ddot{z} = -\frac{\rho g}{m_v} (V_p - \chi z) - \frac{C_d S \rho}{2m_v} |\dot{z}| \dot{z}.$$

Set  $A = \frac{\rho g}{m_v}$  and  $B = \frac{C_d S \rho}{2m_v}$ , we obtain:

$$\ddot{z} = -A (V_p - \chi z) - B |\dot{z}| \dot{z}. \quad (5.5)$$

*Remark 5.2.* The sign of the  $\chi$  coefficient significantly affects the stability of the system. The float is stable for a negative  $\chi$  and unstable for a positive value. Let us take a float that is neutral buoyant for a depth  $z$ . Let us then move the float of  $\delta z$  in the case of a negative  $\chi$ : the variation of volume  $\delta V$  will be of the same sign as  $\delta z$  which will produce a force in the opposite direction of the movement. The float will then go back to its previous depth  $z$ . In the case of a positive  $\chi$  the movement is on the contrary amplified by the variation of volume. Figure 5.5 shows the trajectory over time of two floats with a negative and positive  $\chi$ . The float is stable for  $z = 0$  and was moved of  $\delta z = 0.1$  m. In the case of a negative  $\chi$  the drag force progressively reduces the oscillations while in the case of a positive  $\chi$  the system is clearly unstable and reaches rapidly a constant positive velocity.

## 5.3 Design of a hybrid profiling float

In this section we introduce a new low-cost hybrid profiling float called *SeaBot* that was specifically developed to answer *Challenge 3*. The float is 80 cm long and is designed for shallow water up to 50 m (see Figure 5.6). The whole system uses as much as possible standard and on the shelf mechanical and electronic components. We also tried to limit machining operations for manufacturing the robot.

### 5.3.1 Mechanical architecture

We will give here an overview of the different design problems that have to be considered in order to build a low-cost float. The idea is to give simple and first step design rules.

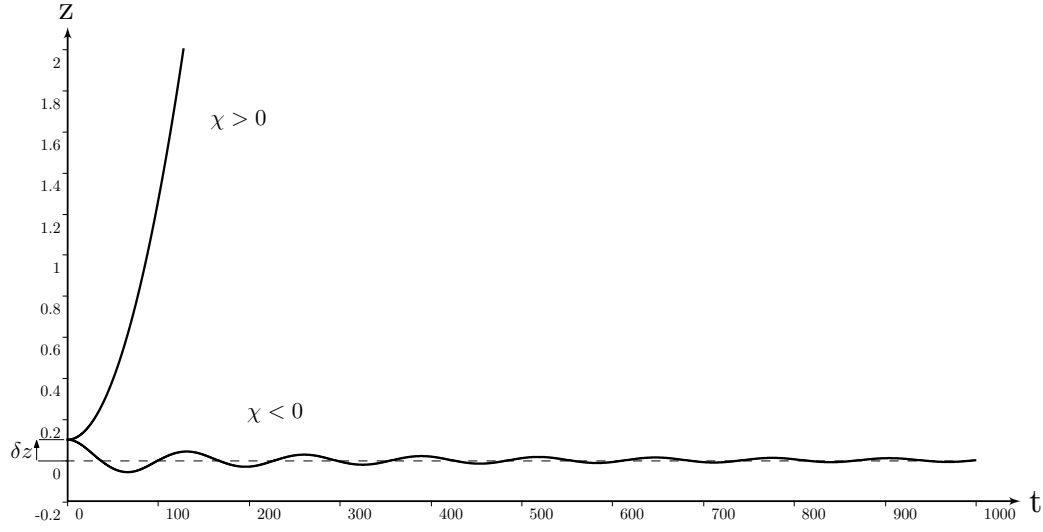


Figure 5.5: Evolution of float depth for a positive and a negative  $\chi$ .

### 5.3.1.1 Float hull

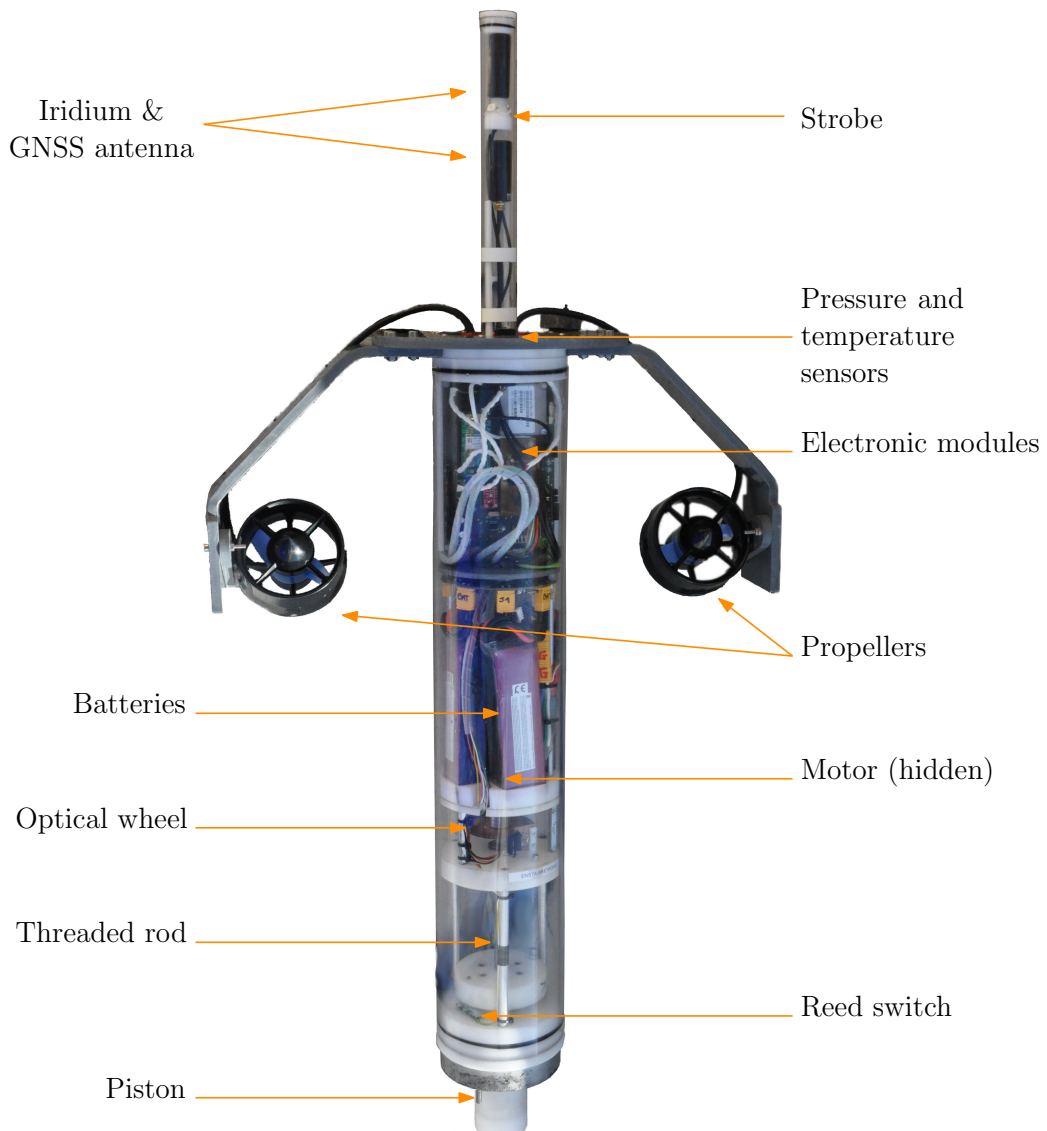
To avoid corrosion phenomenon and to facilitate the development of the float, we have chosen to use a full plastic hull and a transparent pipe. The caps and the piston are in polyoxymethylene (POM-C) and the pipe is in polycarbonate (PC).

To design the thickness of the pipe and the pipe caps, we have to verify that (Agati, Lerouge, and Rossetto, 2008; Aublin et al., 2005):

- $e_{\text{pipe}} > \frac{P \cdot d_{\text{pipe}}}{2\sigma_e}$  where  $e_{\text{pipe}}$  is the thickness of the pipe,  $P$  the external pressure,  $d_{\text{pipe}}$  the diameter of the pipe,  $\sigma_e$  the elastic limit,
- $e_{\text{caps}} > r_{\text{caps}} \sqrt{\frac{2}{3} \frac{P}{\sigma_e}}$  where  $r_{\text{caps}}$  is the radius of the caps.

**Example 5.3.** In the case of the *SeaBot*, we obtain, for a 50 m depth limit and a 120 mm pipe, using Table 5.2,  $e_{\text{pipe}} > 0.4$  mm. We have chosen a thickness of 5 mm that takes into account a safety coefficient and that limits the compressibility issue which will be studied later. We obtain  $e_{\text{caps}} > 8$  mm for the caps.

*Remark 5.4.* Using stainless steel or aluminum would have allowed to reduce the thickness of the pipe to few millimeters but it raises issues with the propagation of wireless signal, with the pipe ovalization at small thickness and with the issue of galvanic corrosion. Moreover stainless steel is more dense than plastic materials, so the gain in thickness should be balanced with the additional mass that reduces the payload mass.

Figure 5.6: The *Seabot* float.

Material	Young modulus $E$ GPa	Elastic limit $\sigma_e$ MPa	Poisson's ratio $\nu$	Density $\rho$ $\text{kg m}^{-3}$
POM-C	2.8	67	0.35	1410
PC	2.3	65	0.37	1200
Stainless steel	190	170	0.3	8000
Aluminum	69	30	0.35	2800

Table 5.2: Approximate values of material mechanical properties

### 5.3.1.2 Float compressibility

Estimating the float compressibility is important to know if the system will be stable or unstable. At a first approximation, we can model the pipe by an infinite cylinder. We know from classic results (Timoshenko, 1941, p.240) that the total radial travel for the external radius of a pressured thick-walled pipe is:

$$u = \frac{1 - \nu}{E} \frac{a^2 P_I - b^2 P_E}{b^2 - a^2} b + \frac{1 + \nu}{E} \frac{a^2 b^2 (P_I - P_E)}{(b^2 - a^2) b}$$

where  $u$  is the total radial travel,  $\nu$  the poisson's ratio,  $E$  the young modulus,  $a$  the internal radius,  $b$  the external radius,  $P_E$  the external pressure and  $P_I$  the internal pressure.

If we neglect the effect of the two caps, the loss of volume can be approximated by:

$$V_{\text{lost}} = \pi(b^2 - (b - u)^2)L$$

where  $L$  is the length of the pipe. We can then deduce an approximation of the float compressibility:

$$\mathcal{K}_f = -\frac{V_{\text{lost}}}{V_f P_E}$$

**Example 5.5.** In the case of the *SeaBot* float, we obtain a mean compressibility for  $P_E = 5$  bar,  $P_I = 0.6$  bar and  $L = 0.6$  m of  $\mathcal{K}_f = 4.30 \times 10^{-9} \text{ Pa}^{-1}$ . We can then deduce the loss of volume per meter  $\chi = 7.22 \times 10^{-7} \text{ m}^3 \text{ m}^{-1}$ . The float is found to be unstable.

The  $\chi_{\text{theory}}$  is similar to the  $\chi_{\text{measured}} \simeq 2.14 \times 10^{-6} \text{ m}^3 \text{ m}^{-1}$ . A more detailed study using numerical simulation should be undertaken to obtain a better estimation of the  $\chi_{\text{theory}}$ . By comparison, a 2 mm thick aluminum pipe would have made the float less compressible than water.

### 5.3.1.3 Auto-ballasting system

The auto-ballasting system (ABS) is based on a 5 cm diameter piston that moves along a M12 steel threaded rod which is rotated with a brushed motor. The position of the piston is given by an optical codewheel (48 counts per revolution) and two reed switches that provide two mechanical zero position and maximum position references. Figure 5.7 shows the mechanical design of the system.

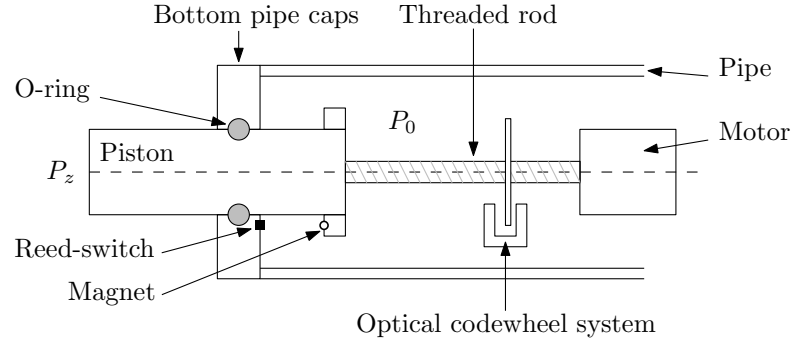


Figure 5.7: Mechanical design of the ABS system.

The required torque that the motor has to deliver can be calculated classically (Agati, Lerouge, and Rossetto, 2008; Aublin et al., 2005) by the following equation:

$$\mathcal{T} = F_P r_{\text{mean}} \tan(i + \varphi) \quad (5.6)$$

where  $\mathcal{T}$  is the torque (in  $N \cdot m$ ),  $F_P$  is the normal force applied on the piston (in  $N$ ),  $r_{\text{mean}}$  the mean radius of the threaded rod (in  $m$ ),  $i$  the thread angle and  $\varphi$  the angle of friction that depends on the materials.

**Example 5.6.** In the *SeaBot* case, we have  $F_P = \rho g z_{\text{max}} S$ ,  $r_{\text{mean}} = 6\text{mm}$ , the screw thread is  $1.75\text{mm}$  which gives  $i = 8^\circ$  and if we assume a friction coefficient of  $0.4$  between steel and POM-C,  $\varphi = 22^\circ$ . We obtain  $\mathcal{T}_{\text{max}} = 3.4\text{ N m}$ .

The *SeaBot* motor (MFA Como 970D1561) is a  $\mathcal{P}_m = 19.8\text{ W}$ , of  $0.015\text{ N m}$  at maximum efficiency and  $0.1\text{ N m}$  at the maximum torque, with a  $156:1$  reduction gearbox, which gives a maximum output torque between  $2.34\text{ N m}$  and  $15.6\text{ N m} > \mathcal{T}_{\text{max}}$ . The output rotation speed is  $93\text{RPM}$  so we can compute the theoretical maximum volume variation of the piston per time:  $\dot{V}_p = 5.32 \times 10^{-6}\text{ m}^3\text{ s}^{-1}$ .

A compromise has to be found between the diameter of the piston, the maximum torque of the motor and the maximum volume rate of the piston.

#### 5.3.1.4 Additional systems

The *SeaBot* float is also equipped with two low-cost thrusters that can be used at surface to correct its position.

In addition, the internal part of the float is maintain at a pressure of  $600\text{ mbar}$ . This vacuum holds the two caps and the O-rings. This also provides an easy way to detect important leak issues.

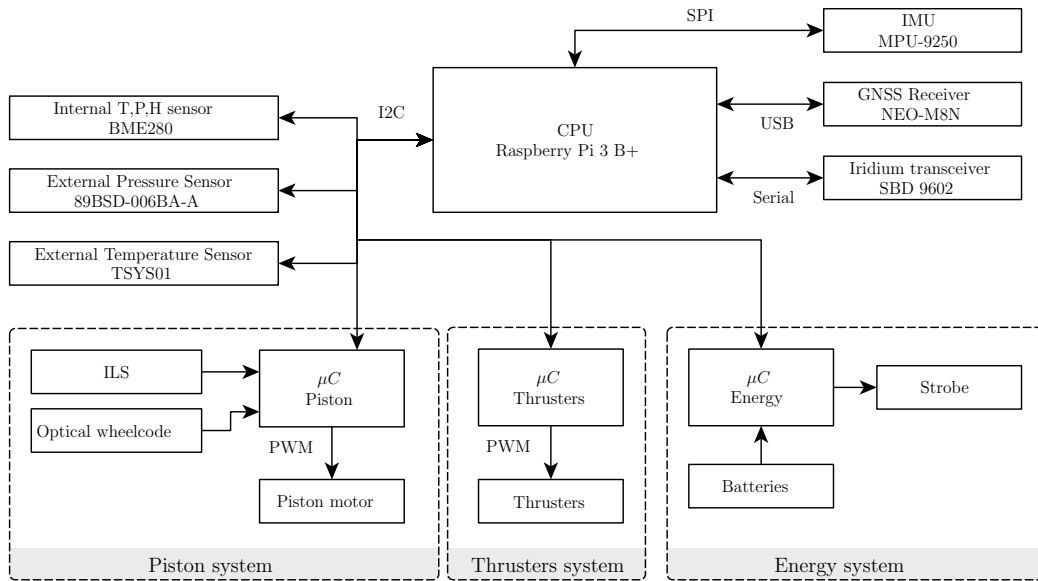


Figure 5.8: Electronic design.

### 5.3.2 Electronic architecture

The main electronic is based on a Raspberry Pi 3 B+ board and microcontrollers ( $\mu C$ ) dedicated to real time control and hardware interfaces.

The float is equipped with a 18 cm accuracy, 0.1 mm resolution pressure sensor, an external temperature sensor, a MEMS IMU, a GNSS receiver, an Iridium transceiver, an optical codewheel for the piston and an internal temperature, pressure and humidity sensor to detect water leak issues. Figure 5.8 shows the electronic architecture.

*Remark 5.7.* Monitoring the humidity level appears to be more efficient to detect small leaks than monitoring the internal pressure which requires important water ingress to change.

The float has four 5Ah 3S LiPo batteries that provide a total of 20Ah. This gives around 226 Wh which means about one day autonomy.<sup>6</sup> The electronic system without the motor and thrusters consumes around 2.5 W. The electronic energy consumption could be greatly optimized in a second step design.

<sup>6</sup>This depends heavily on the use of thrusters and the dive curve.

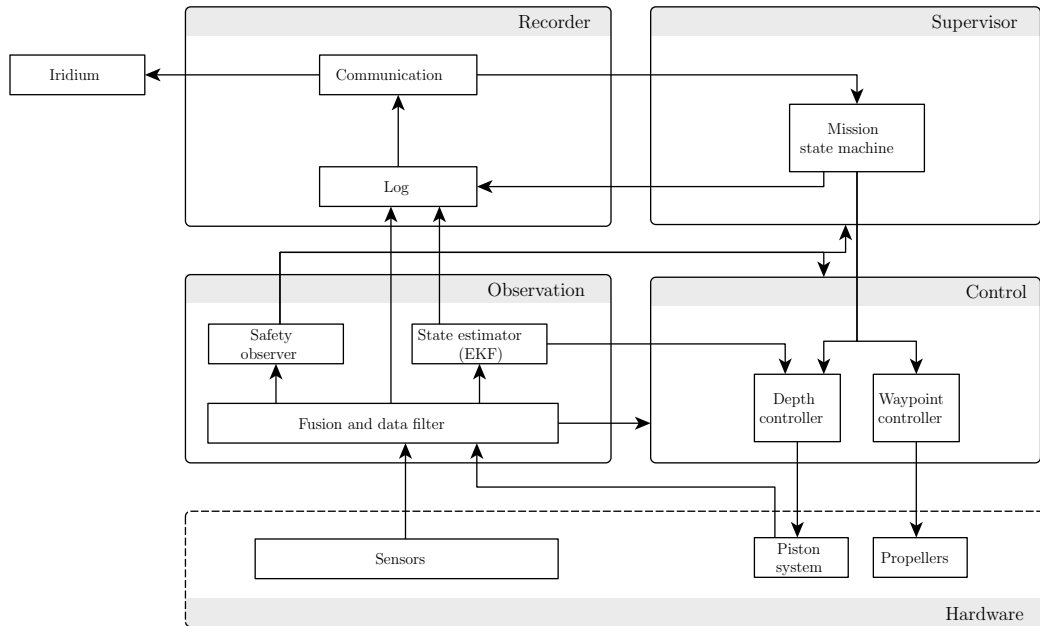


Figure 5.9: Software functional architecture.

### 5.3.3 Software architecture

The software is based on the ROS middleware. It is built around nodes that handle specific tasks and can communicate with each other. The *SeaBot* source code is available online (see Section 1.3 on page 13). Figure 5.9 shows a simplified functional architecture of the software. Note that the robot implements a safety observer that constantly checks the parameters of the float (maximum depth, leak issue, etc.) and triggers an emergency surfacing if necessary.

## 5.4 Depth controller

In this section, we will deal with the depth controller of the float during diving. A key point, in the context of low-cost actuators and sensors, is to implement an efficient control law that minimizes the energy consumption while maintaining a low error relative to the depth set point.

Several approaches have been used in the literature: a survey of profiling float controllers can be found in (Shi et al., 2017). Classic PID based controllers are not suitable in the context of low-cost floats as underlined in (McGilvray and Roman, 2010) because of the time required to tune experimentally their coefficients. State of the art float controllers now use state

feedback (Bessa et al., 2017) or adaptive control (Berkenpas et al., 2018) techniques. The main difficulty of those controllers is the ability to know an accurate dynamical model of the robot. Indeed, several parameters such as the buoyancy depend on the surrounding water properties. An online estimator must therefore be implemented. To solve the problem, several techniques have been used including fuzzy inferences (Bessa et al., 2017) and full state observers (McGilvray and Roman, 2010).

In the following section, we will propose a new method built on a state feedback controller and on an online estimator based on an Extended Kalman Filter (EKF). This new method better takes into account the compressibility of the float. It also drives the float with a velocity constraint which is new. Indeed, from a theoretical point of view, the energy loss of the system is mainly due to drag forces which are linked to the velocity of the float. A precise control of the float velocity is therefore crucial.

The float system can be modeled through the following equation:  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$  where  $\mathbf{x}$  is the state vector of the system,  $\mathbf{f}$  the evolution function of the system and  $\mathbf{u}$  the input. From Equation (5.5), we obtain:

$$\dot{\mathbf{x}} = \begin{pmatrix} \ddot{z} \\ \dot{z} \\ \dot{V}_p \end{pmatrix} = \begin{pmatrix} -A(V_p - \chi z) - B|\dot{z}|\dot{z} \\ \dot{z} \\ u \end{pmatrix} \quad (5.7)$$

where  $u$  is the piston volume rate.

### 5.4.1 Control law

In a context of low energy consumption, we want to avoid as much as possible any overshoot of the control which would cause unnecessary movements of the piston. In terms of energy, the mechanical work of the piston depends of the velocity and of the loss of volume per meter  $\chi$ . If  $\chi = 0$ , the float can move from an equilibrium depth position to an other with an  $\epsilon$  move of the piston: the work is then directly link to the velocity of the movement and not to the traveled depth.

To be able to limit the velocity while reaching the desired depth, we chose to control the float with a vector field that links the velocity and the depth error to the depth set-point (see Figure 5.10):

$$\dot{z} = \beta \arctan\left(\frac{\bar{z} - z}{\alpha}\right)$$

where  $\bar{z}$  is the depth set-point and  $(\alpha, \beta)$  is a pair of constant parameters. Other functions such as sigmoids could have been chosen as long as they



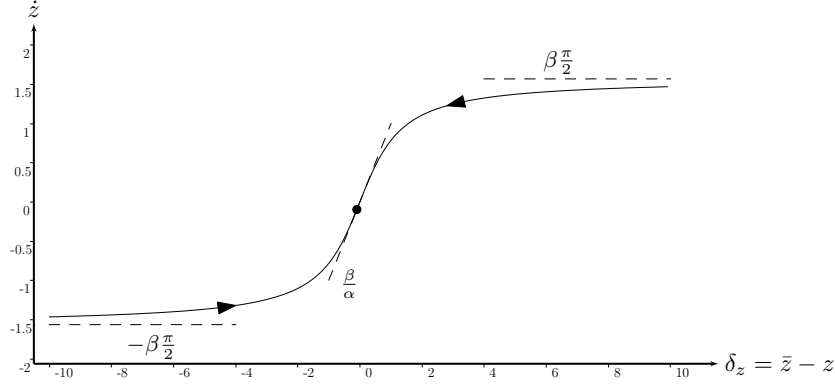


Figure 5.10: Map of the set-point velocity as a function of the depth error  $\delta_z$

are smooth, as this is required to apply state feedback techniques. The coefficients  $\alpha$  and  $\beta$  will be chosen depending on the performances required by the user application in particular the maximum velocity  $\dot{z}_{\max} = \beta \frac{\pi}{2}$  and the deceleration phase near the depth set-point though  $\frac{\beta}{\alpha}$ . The adjustment of these parameters will be discuss in Section 5.5. Unless stated otherwise, we will let  $\alpha = 1$ .

To apply state feedback linearization technique (Jaulin, 2015), we chose for the system output  $y = \dot{z} - \beta \arctan\left(\frac{\bar{z}-z}{\alpha}\right)$ . Our system has a relative degree of 2 which requires to derive two times the output.

$$\dot{y} = \ddot{z} - \frac{\beta}{\alpha} \frac{-\dot{z}}{1+e^2} = \ddot{z} + \gamma \frac{\dot{z}}{D}$$

where  $e = \frac{1}{\alpha}(\bar{z} - z)$ ,  $D = 1 + e^2$ ,  $\gamma = \frac{\beta}{\alpha}$ ,

$$\ddot{y} = \dddot{z} + \gamma \frac{\ddot{z}D - \dot{z}\dot{D}}{D^2} = \ddot{z} + \gamma \frac{\ddot{z}D + 2\alpha^{-2}e\dot{z}^2}{D^2}$$

as  $\dot{D} = -2\alpha^{-2}e\dot{z}$  and with  $\ddot{z} = -A(u - \chi\dot{z}) - 2B|\dot{z}|\ddot{z}$ ,

$$\ddot{y} = -Au + A\chi\dot{z} - 2B|\dot{z}|\ddot{z} + \gamma \frac{\ddot{z}D + 2\alpha^{-2}e\dot{z}^2}{D^2}.$$

We then chose  $u$  such that  $y$  is solution of  $\lambda_3\ddot{y} + \lambda_2\dot{y} + \lambda_1y = 0$  where  $\lambda_1, \lambda_2, \lambda_3$  are constant coefficients. In order to avoid any overshoot we want a single negative pole  $s$  such that the characteristic equation of the previous equality is  $(1 - s)^2$ , which gives the coefficient values:  $\lambda_3 = 1$ ,  $\lambda_2 = -2s$  and  $\lambda_1 = s^2$ . The control law can be then expressed as:

$$u = \frac{1}{A}(-2s\dot{y} + s^2y + \gamma \frac{\ddot{z}D + 2\alpha^{-2}e\dot{z}^2}{D^2} - 2B|\dot{z}|\ddot{z}) + \chi\dot{z}.$$

This allows  $y$  to converge towards 0 at a speed of  $\sim e^{st}$ . The pole  $s$  should be chosen in function of the dynamic of the system.

*Remark 5.8.* The previous model is only valid when the float is completely immersed. This is not the case at surface when antennas are emerged. This is why a simple finite-state machine switches between a simple sinking procedure that slowly retracts the piston until a certain depth  $z_f$  is reached, and a state feedback controller that is activated below this depth.

## 5.4.2 Estimation of unknown parameters

The main issue, with the control law described above, is that the exact volume  $V_p$  of the piston and the  $\chi$  parameter are unknown. Concerning the volume, we measure with a high precision the volume of the piston  $V_m$  from a mechanical zero reference but we do not know the offset  $V_o$  such that the float is at equilibrium for a zero depth ( $V_p = V_m + V_o$ ). The parameter  $\chi$  is even more complex to estimate as it also depends of surrounding water properties.

This is why an EKF will be used to estimate  $V_o$  and  $\chi$ . Note that some of the modeling errors would be also compensated by the estimation of both variables. By using Equation (5.7), we can obtain a specific system for the estimation of  $V_o$  and  $\chi$ . Note that  $V_p$  is here considered as the input  $u$ . We measure  $z$  and we assume that  $V_0$  and  $\chi$  are constant over time. With the state vector  $\mathbf{x} = (\dot{z}, z, V_0, \chi)^\top$ , we have for the continuous system:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}_c(\dot{\mathbf{x}}, u) & = \begin{pmatrix} -A(u - \chi z) - B|\dot{z}|\dot{z} \\ \dot{z} \\ 0 \\ 0 \end{pmatrix} \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) & = (z) \end{cases} \quad (5.8)$$

We recall the Kalman prediction and corrector equations<sup>7</sup>, in the case of a discrete time system, with an euler integration scheme at step  $k$ , and a  $dt$  duration between steps:

- Prediction

$$\begin{cases} \hat{\mathbf{x}}_{k+1|k} = \mathbf{f}(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k) = \hat{\mathbf{x}}_{k|k} + dt \cdot \mathbf{f}_c(\mathbf{x}_k, \mathbf{u}_k) & \text{(predicted estimation)} \\ \Gamma_{k+1|k} = \mathbf{A}_k \cdot \Gamma_{k|k} \cdot \mathbf{A}_k^\top + \Gamma_{\alpha_k} & \text{(predicted covariance)} \end{cases}$$

<sup>7</sup>notations are taken from (Jaulin, 2015)

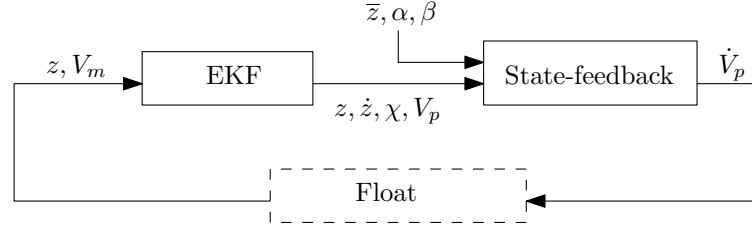


Figure 5.11: Depth controller structure.

- Update

$$\begin{cases} \hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{z}}_k & \text{(corrected estimation)} \\ \Gamma_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{C}_k) \Gamma_{k|k-1} & \text{(corrected covariance)} \\ \tilde{\mathbf{z}}_k = \mathbf{y}_k - \mathbf{C}_k \hat{\mathbf{x}}_{k|k-1} & \text{(innovation)} \\ \mathbf{S}_k = \mathbf{C}_k \Gamma_{k|k-1} \mathbf{C}_k^\top + \Gamma_{\beta_k} & \text{(covariance of the innovation)} \\ \mathbf{K}_k = \Gamma_{k|k-1} \mathbf{C}_k^\top \mathbf{S}_k^{-1} & \text{(Kalman gain)} \end{cases}$$

where

- $\mathbf{A}_k = \frac{\partial \mathbf{f}(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k)}{\partial \mathbf{x}} = \begin{pmatrix} -2B \left| \hat{z} \right| & A\hat{\chi} & -A & A\hat{z} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot dt + \mathbf{I}_{4 \times 4}$  is the evolution matrix,

- $\mathbf{C}_k = \frac{d\mathbf{g}(\hat{\mathbf{x}}_{k|k-1})}{d\mathbf{x}} = (0 \ 1 \ 0 \ 0)$  is the observation matrix,

- and  $\Gamma_\alpha, \Gamma_\beta$  are respectively the process and the observation noise covariance matrices. In our case we set them diagonal and constant. Their coefficients depend on the sensors accuracy and of the dynamic of the float.

Figure 5.11 illustrates the depth controller structure with the EKF and the state-feedback.

### 5.4.3 Experimental results

**Tests in a controlled environment** The float system was tested in a 20 m deep sea water basin at Ifremer<sup>8</sup> Brest (see Figure 5.12). The state feedback controller and the EKF were running at 5 Hz with a transition depth  $z_f =$

<sup>8</sup>Institut Français de Recherche pour l'Exploitation de la Mer

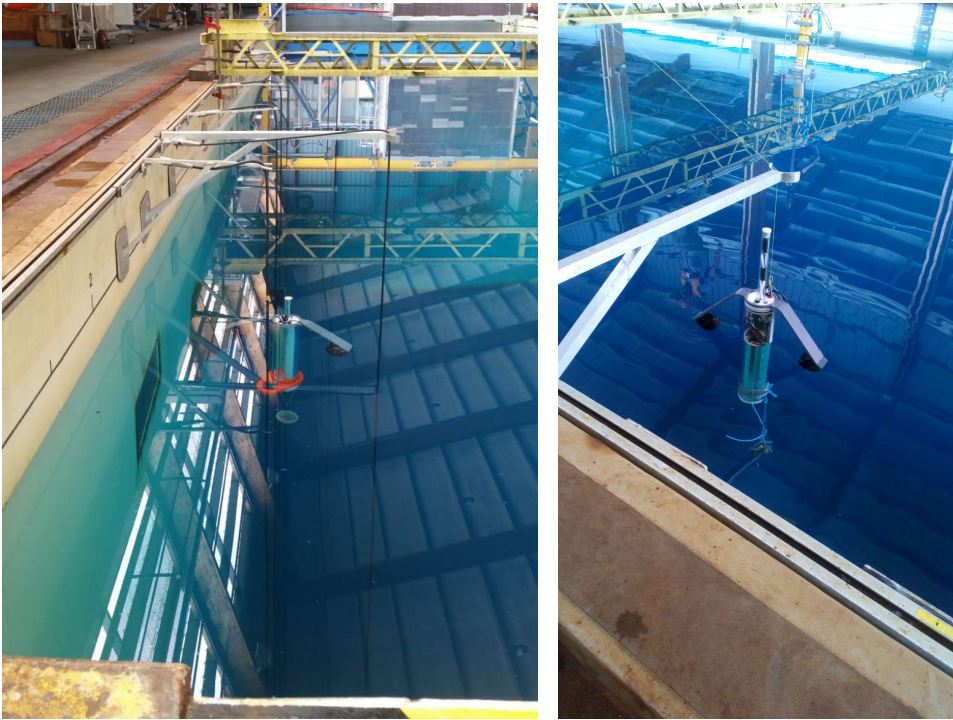


Figure 5.12: Ifremer Brest water tank with the *SeaBot* float. The robot is secured along a rope guide.

0.3 m. The mission was to reach five different depth levels  $\{1, 5, 10, 15, 18\}$  m with a maximum speed of  $|\dot{z}_{\max}| = 0.04 \text{ m s}^{-1}$ ,  $\alpha = 1$  and  $s = -1$ . Figure 5.13 shows the trajectory over time and Figure 5.14 shows the piston volume measured  $V_m$ . There is no overshoot of the control and we clearly see, on the volume  $V_m$  plot, the compensation of the loss of volume: the deeper the robot is, the larger the volume of the piston output must be.

**Evolution of  $\chi$**  Measuring the volume of the piston once the float is stabilized at every depth level gives an idea of the value of  $\chi$ . From experimental data, the loss of volume appears not to be linear with depth  $z$  but quadratic with respect to  $z$ . However, the EKF handles this model error and adjust the value of  $\chi$  and  $V_0$  (see Figure 5.15): this is why  $\chi$  is not constant over time.<sup>9</sup>

<sup>9</sup>The deformation of the pipe is not constant but looks like an hourglass. Indeed, the caps at the end prevent the pipe to be compressed regularly.



Figure 5.13: Depth  $z$  (in meter) of the float in function of time  $t$  (in seconds). The setpoint depth trajectory is in red and the float depth trajectory is in black.

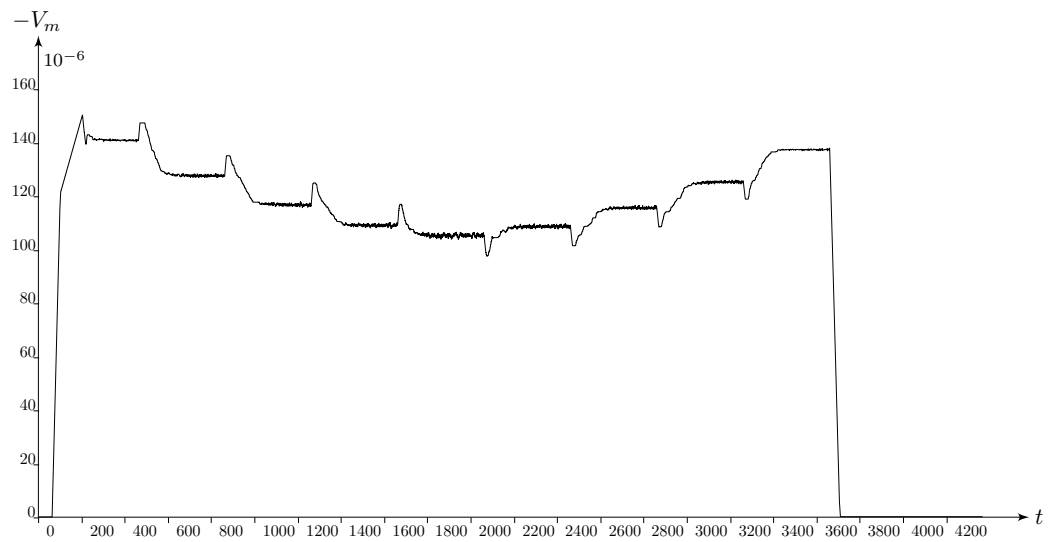
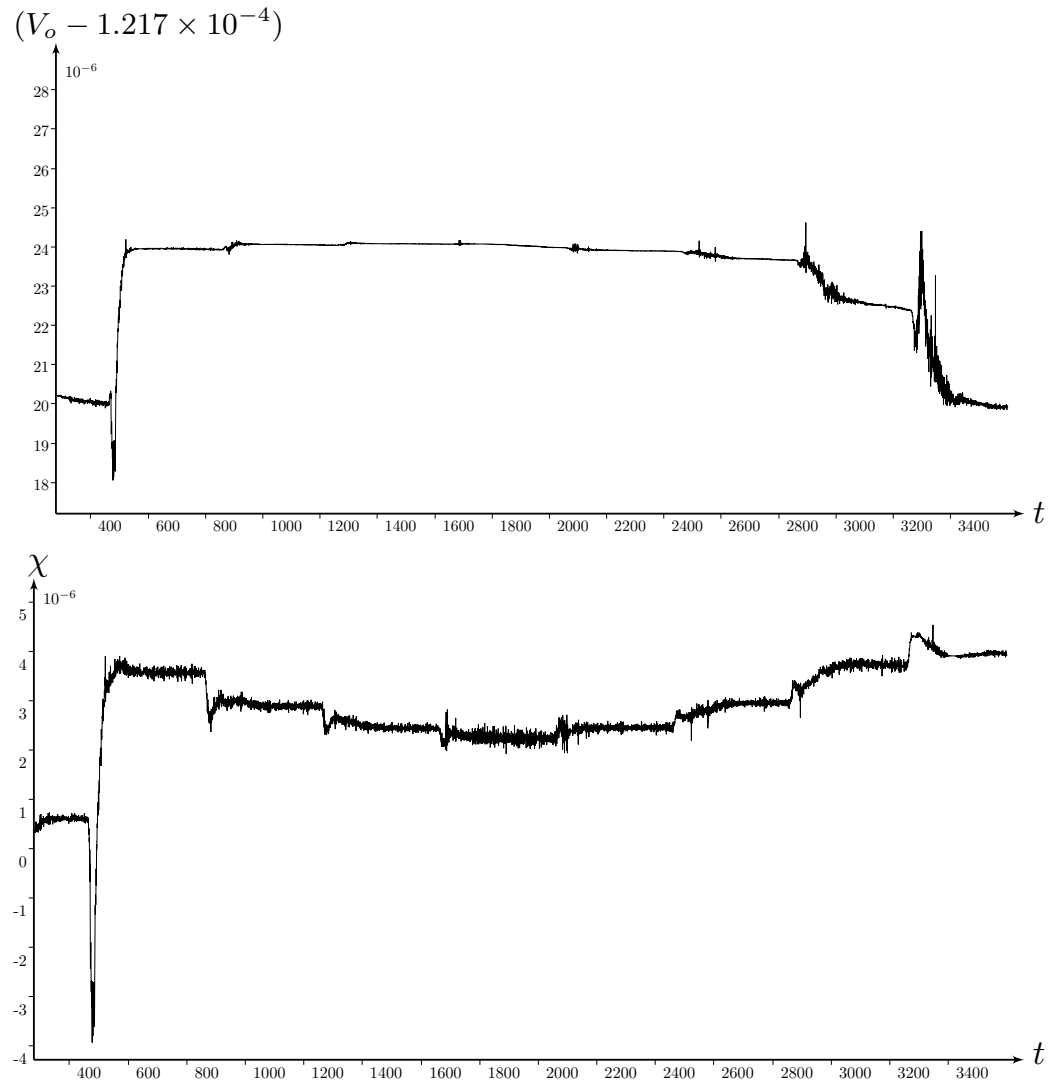


Figure 5.14: Piston volume measured  $V_m$  in function of time  $t$  (in seconds)

Figure 5.15: Estimation of  $V_0$  and  $\chi$  by the EKF over time

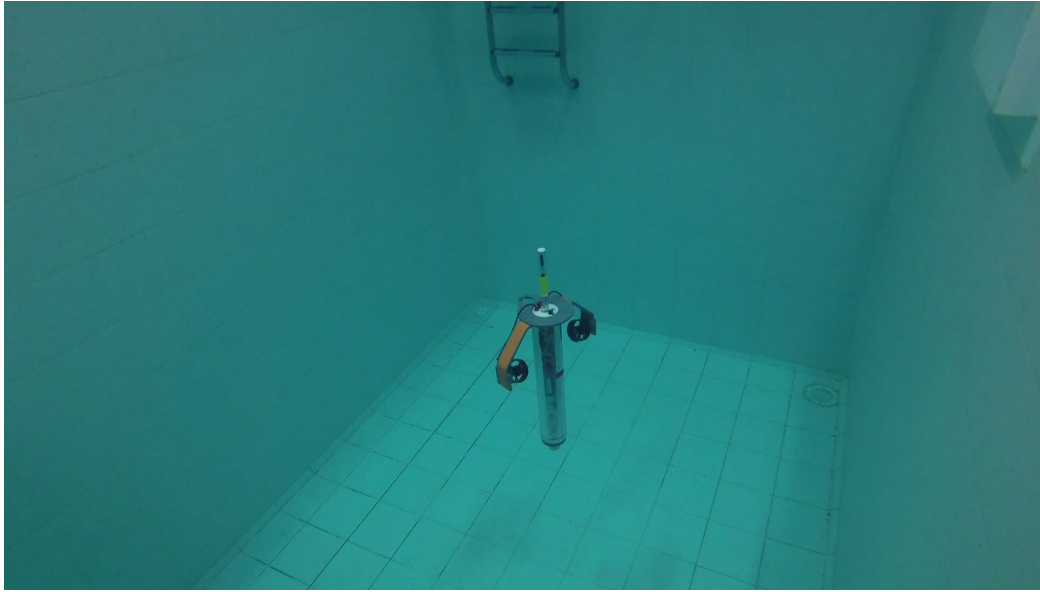


Figure 5.16: Last test of the *Seabot* depth controller in the ENSTA Bretagne 2.5m deep pool before a deployment in the Bertheaume cove. Note that we added colored stripes to make it easier to find the robot at the end of the mission.

**Energy consumption** To reduce the energy consumption during ascending and descending phases, we compute an interval of input  $[u]$  for an interval of maximum velocity  $[\dot{z}_{\max}]$  and we choose the input that minimizes  $|u|$ . A no piston movement strategy could also be adopted when the set-point is reached in the case of a stable float but this is not the case for our system.

**Depth error** For this first test, the depth error after the depth set-point is reached is of few centimeters (in most cases under 2 cm). Some depth bias of up to 4 cm can be observed which could come from mechanical hysteresis or error in the model. Adding an integral effect to the control law is a solution to compensate these small bias.

**Tests in real environments** Ocean trials have been conducted in the Brest bay (see Figure 5.17), in the Bertheaume cove (see Figure 5.16) and in the Guerlédan lake (see Figure 5.18). Results are the subject of ongoing analysis at the time these lines are written concerning sea trials.

The Guerlédan lake mission consisted of 8 identical two-hours series. The robot had to perform 20-minute depth stops at:  $\{10, 17.5, 25, 17, 9.5, 2, 0\}$  m for a total of 17 hours mission. The mission started on the 10<sup>th</sup> of October



Figure 5.17: Two *SeaBot* before their mission in the bay of Brest.



Figure 5.18: A *Seabot* after ending its mission in the lake of Guerlédan. We can only see the yellow antenna emerging from the surface at the center of the picture. An air drone is filming the scene.



Parameter	Lower bound	Upper bound
$[V_p]$	$-2.148 \times 10^{-5} \text{ m}^3$	$1.503 \times 10^{-4} \text{ m}^3$
$[\dot{V}_p]$	$-1.431 \times 10^{-6} \text{ m}^3 \text{ s}^{-1}$	$1.431 \times 10^{-6} \text{ m}^3 \text{ s}^{-1}$

Table 5.3: *Seabot* piston parameters (experimental results,  $\dot{V}_p$  is limited by software compare to the maximum possible values)

2019 at 15h06UTC and lasted all the night. Figure 5.19 shows the result with a zoom on one of the stop. We can see that the depth error is relatively low with an order of magnitude of few millimeters.<sup>10</sup>

We can also note that just after 9 hours of mission, the robot encountered wrong pressure data that led to a return to the surface. As a result of this event, the EKF diverged to false values which explains why there are small overshoots on the next depth stops. The situation came back to normal around 11 hours of mission. Additional safety checks were added to prevent this problem from happening again.

For each surface rise, the robot sent its position through a satellite connection and used its thrusters to reach a next predefined area to dive. We will not provide additional data concerning the surface travel as the results were not enough convincing. Indeed, the robot encountered failures of its magnetic compass which should be corrected in future experiments.

## 5.5 Validation of the depth control law

In this section, we will discuss how we can use the tools of previous chapters to validate the depth control law of the *SeaBot*. We will first consider the open-loop system, then we will add the control law to deal with the closed-loop system. Finally we will deal with advance validation issues.

The *SeaBot* has mechanical constraints which bounds the volume  $V_p \in [V_p]$  and the volume rate of the piston  $\dot{V}_p \in [\dot{V}_p]$ . These constraints narrow the  $\alpha, \beta, s$  and  $\bar{z}$  possible values. The parameters for the *SeaBot* piston are summarized in Table 5.3. The asymmetry between  $V_p^+$  and  $V_p^-$  is due to the need to have a sufficient reserve buoyancy to emerge antennas.

### 5.5.1 Open-loop float performances

In this section, we will study the performance of the float in regards of maximum reachable depths and velocities.

<sup>10</sup>The implementation of the controller was improved compared to the first test.

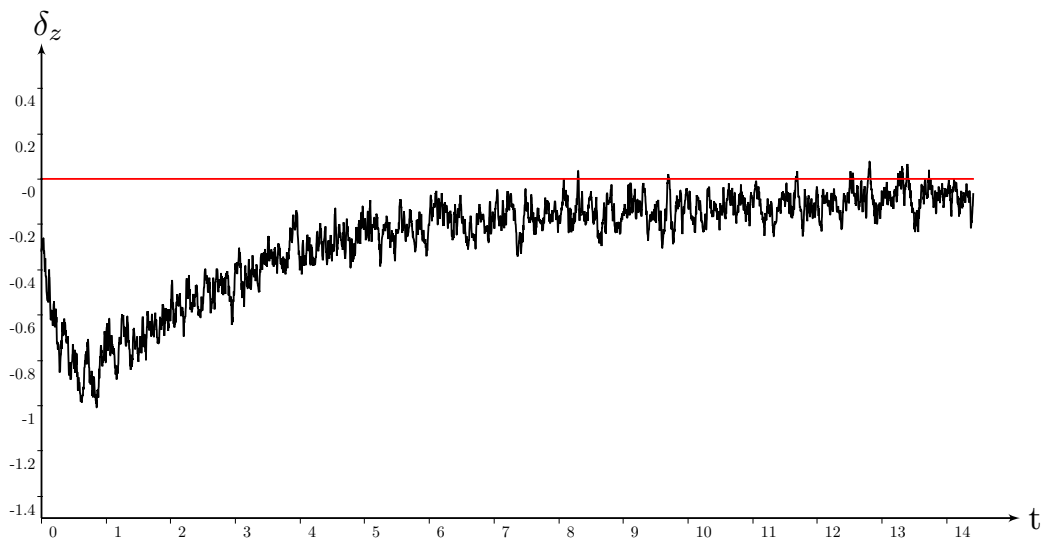
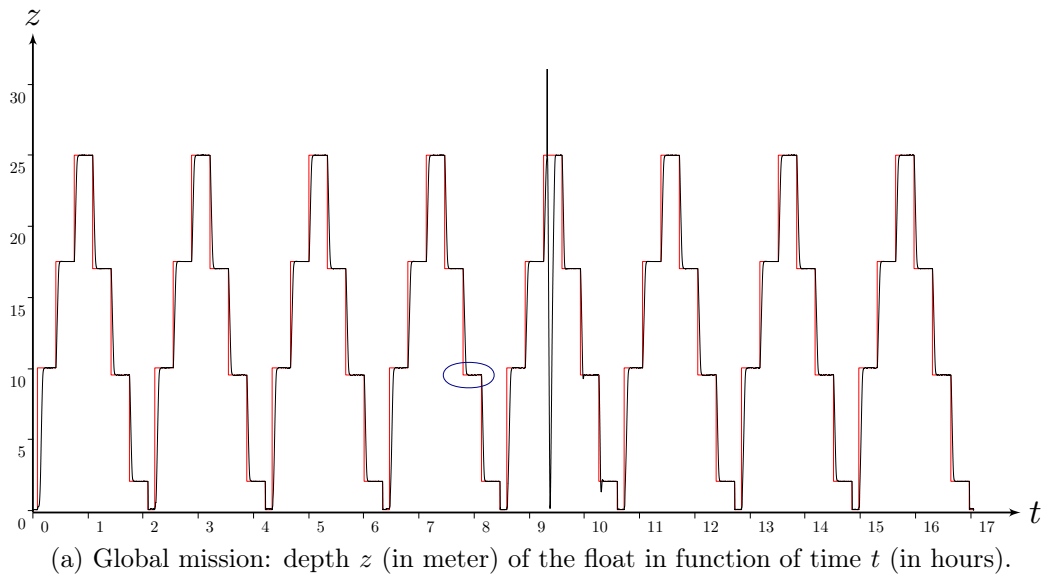


Figure 5.19: The Guerlédan lake mission. The depth set-point trajectory is in red and the float depth trajectory is in black.

The maximum depth is not only limited by the hull durability against pressure but also by the loss of volume due to compressibility. It is equal<sup>11</sup> to around 70 m. We have set a practical depth limit to  $z_{\max} = 50$  m to keep a safety margin which covers the mechanical strength of the pipe and the compressibility issue. Similarly, the float cannot exceed a certain velocity  $\dot{z}_{\max}$  due to compressibility.<sup>12</sup>

**Largest viable set** To highlight the depth and velocity limit, we will consider a simplified model of the float where the volume of the piston can be set instantaneously, *i.e.*  $\dot{V}_p$  is not limited and  $u = V_p$ . This allow to work in a 2D space as algorithms of Chapter 3 are not fully implemented in 3D. We have the following simplified system with the state vector  $\mathbf{x} = (\ddot{z}, \dot{z})^\top$ :

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} -A(u - \chi z) - B|\dot{z}|\dot{z} \\ \dot{z} \end{pmatrix}. \quad (5.10)$$

An interesting set to bracket is  $\text{Viab}_{\mathbf{f}}(\mathbb{X}, \mathcal{U})$  where we choose  $\mathcal{U} = \{\mathbf{u}(\cdot) \mid \forall t, \mathbf{u}(t) \in [V_p]\}$  and  $\mathbb{X} = [-20, 80] \times [-0.4, 0.4]$ . This set will give the limitation of performance of the float. Indeed, it represents the set of all states such that, there exists an input, such that the robot will stay and was forever in  $\mathbb{X}$  (positive and negative time). Figure 5.20 gives the viability kernel<sup>13</sup> of the simplified system (see Section 4.1 for explanations on how the set is computed).

The depth and velocity limitations and their link clearly appear in the result. We also see that there is a difference of velocity between the case where the float is at the surface and at the maximum depth. This can be explained by the fact that, at surface, there is more positive reserve buoyancy than negative. We can see that, at the maximum depth, the velocity is equal

<sup>11</sup>At equilibrium in the case of the maximum depth, we have from Equation (5.5):  $V_p - \chi z_{\max} = 0$  which implies that  $z_{\max} = \max_{V_p \in [V_p]} \left( \frac{V_p}{\chi} \right) = 70$  m. A stable float ( $\chi < 0$ ) will not be able to go deeper whereas an unstable float ( $\chi > 0$ ) will not be able to go back to a lower depth. Indeed, it will not have enough reserve buoyancy to come back to surface, if it goes deeper than this limit.

<sup>12</sup>If we suppose that the float has stabilized its velocity to  $\dot{z}_{\max}$ , we then have from Equation 5.5:  $-A(V_p - \chi z) - B|\dot{z}_{\max}|\dot{z}_{\max} = 0$  which implies that

$$\dot{z}_{\max|z} = \left( \sqrt{\left| \frac{A}{B} (V_p - \chi z) \right|} \right)_{V_p \in \{V_p^-, V_p^+\}}. \quad (5.9)$$

<sup>13</sup>The drag coefficient was not estimated accurately so the results only give an order of magnitude.

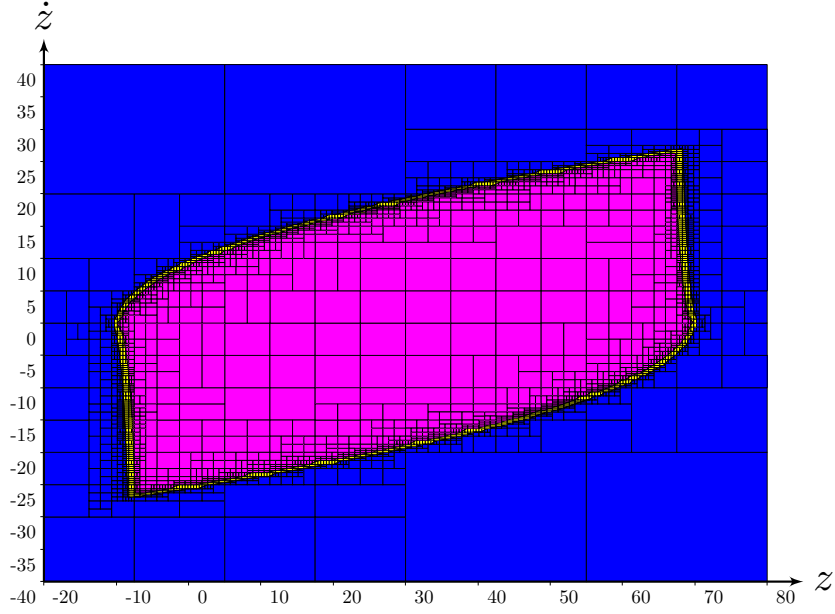


Figure 5.20: Largest viable set  $\text{Viab}_f(\mathbb{X}, \mathcal{U})$  for the *SeaBot* float. Velocity is in  $\text{mm s}^{-1}$  and depth in m.

to zero. Therefore, this depth should not be reached as it will not be possible to come back at lower depths afterwards. The paths generated by the control law should keep the profiling float inside  $\text{Viab}_f(\mathbb{X}, \mathcal{U})$  to avoid any issue.

**Largest positive viable set** A second interesting set to bracket is  $\text{Viab}_f^+(\mathbb{X}, \mathcal{U})$ , which corresponds to all the states where the float can be released in  $\mathbb{X}$  such there exists an input, such that it will stay forever (positive time) in  $\mathbb{X}$ . Figure 5.21 shows the result. If an input brings the float out of  $\text{Viab}_f^+(\mathbb{X}, \mathcal{U})$  then it will never be able to return to  $\mathbb{X}$  and therefore to  $\text{Viab}_f(\mathbb{X}, \mathcal{U})$ .

**Piston velocity limitation** We consider now the limitation induced by  $\left[\dot{V}_p\right]$ . The speed of the loss of volume due to compressibility is linked to the velocity of the float. Therefore, the piston should move fast enough to compensate this loss. Otherwise, in the case of an unstable float, it will not succeed in decelerating. In the stable case, the velocity will be limited by:  $\dot{z}_{\max} = \frac{\dot{V}_p}{\chi} \mid_{\dot{V}_p \in \{\dot{V}_p^-, \dot{V}_p^+\}} \in \{-0.66, 0.66\} \text{ m s}^{-1}$ . Therefore  $\left[\dot{V}_p\right]$  limits the maximum allowable velocity of the float.

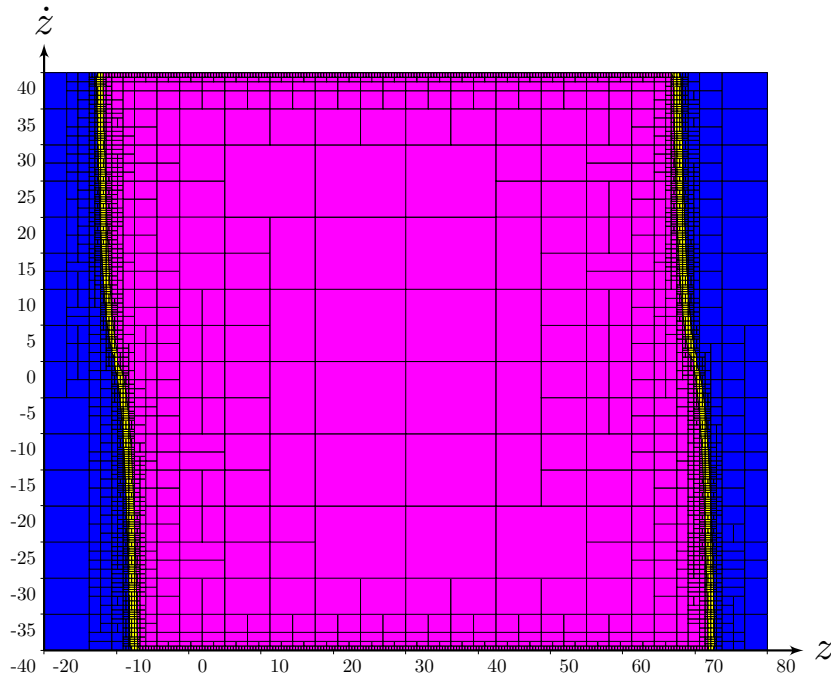


Figure 5.21: Largest positive viable set  $\text{Viab}_f^+(\mathbb{X}, \mathcal{U})$  for the *SeaBot* float. Velocity is in  $\text{mm s}^{-1}$  and depth in m.

### 5.5.2 Closed-loop system

We now consider the closed-loop system with the depth controller. To keep a 2D system, we can, as described in Section 5.4.1, compute a state feedback control for the simplified system (5.10):

$$u = \frac{1}{A} \left( \lambda (\beta \arctan(e) - \dot{z}) - \left( B |\dot{z}| + \frac{\gamma}{D} \right) \dot{z} \right) + \chi z$$

where we set  $\lambda = 10$ . We also know that the piston has mechanical volume limits which means that  $u$  is saturated (see Table 5.3 on page 181). We will only consider a saturated system hereafter.

**Sensors uncertainties** The first problem we will consider is to bracket the set of the state space for which the closed-loop system is safe, *i.e.* it stays forever inside a set  $\mathbb{X}$ . We consider here a system with uncertainties on the sensors of the form  $\dot{\mathbf{x}} = \mathbf{f}_w(\mathbf{x} + \mathbf{w})$  where  $\mathbf{w} \in \mathcal{W}$  is the noise of the sensors. We set  $\mathcal{W} = [-1e^{-3}, 1e^{-3}] \times [-5e^{-3}, 5e^{-3}]$ , the targeted depth  $\bar{z} = 15$  and  $\mathbb{X} = [-0.015, 0.015] \times [14.7, 15.3]$ .

We first bracket the set  $\text{Inv}_{\mathbf{F}_w}^+(\mathbb{X})$  where we consider the differential inclusion  $\mathbf{F}_w$ . We will not show the result in a figure as nearly all the states of

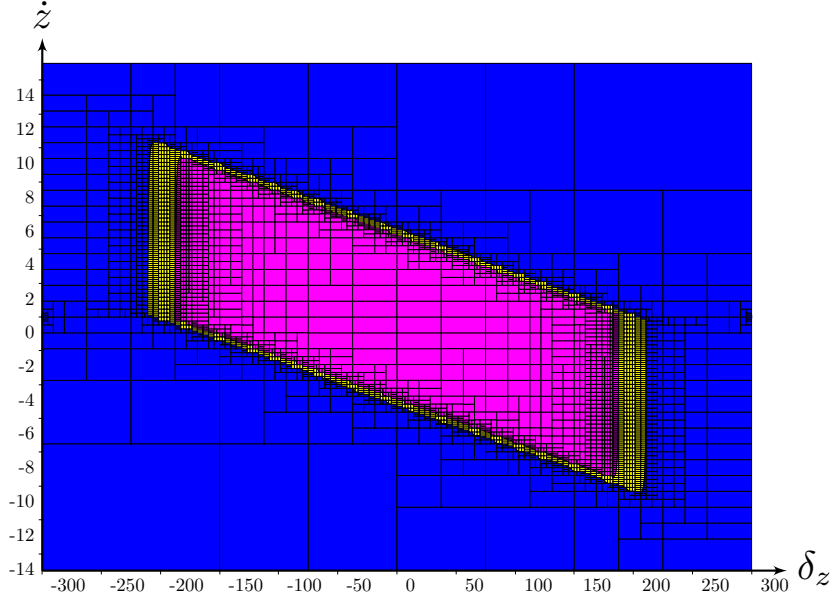


Figure 5.22: Largest negative viable set  $\text{Viab}_{\mathbf{f}_w}(\mathbb{X}, \mathcal{W})$  for the *SeaBot* float. Velocity ( $\text{mm s}^{-1}$ ) in function of the depth error (mm).

$\mathbb{X}$  are inside  $\text{Inv}_{\mathbf{F}_w}^+(\mathbb{X})$  which produces an almost all magenta figure. Therefore, almost every states in  $\mathbb{X}$  will stay forever inside  $\mathbb{X}$ , except for few states near the border of  $\mathbb{X}$ . These states belong, in fact, to paths that re-enter in  $\mathbb{X}$  after escaping from it. For all initial states that belong to  $\text{Inv}_{\mathbf{F}_w}^+(\mathbb{X})$  and for all  $w$ , the path will stay forever in the future inside  $\mathbb{X}$ .

The second set we will bracket is  $\text{Viab}_{\mathbf{f}_w}(\mathbb{X}, \mathcal{W})$ . The result is given in Figure 5.22. We can conclude that if the system is initialized inside  $\text{Inv}_{\mathbf{F}_w}^+(\mathbb{X}) \supset \text{Viab}_{\mathbf{f}_w}(\mathbb{X}, \mathcal{W})$ , it will not be able to reach, in the worst case, a set of final states smaller than  $\text{Viab}_{\mathbf{f}_w}(\mathbb{X}, \mathcal{W})$ .

**Set-points** A second interesting problem is to consider the depth set point as the input and to study the behavior of the closed-loop system. We have a system of the form  $\dot{\mathbf{x}} = \mathbf{f}_{\bar{z}}(\mathbf{x}, \bar{z})$  where  $\bar{z} \in \mathcal{Z}$  is the set of depth set-points. Let us take  $\mathcal{Z} = [0, 50]$ .

We can bracket the set  $\text{Viab}_{\mathbf{f}_{\bar{z}}}(\mathbb{X}, \mathcal{Z})$  as shown in Figure 5.23. Similarly to the previous section we have:  $\text{Viab}_{\mathbf{f}_{\bar{z}}}^-(\mathbb{X}, \mathcal{Z}) \subset \text{Inv}_{\mathbf{F}_{\bar{z}}}^+(\mathbb{X})$ . We can conclude that if the system is initialized in  $\text{Inv}_{\mathbf{F}_{\bar{z}}}^+(\mathbb{X})$ , it will not be able, in the worst case, to evolve in a smaller set than  $\text{Viab}_{\mathbf{f}_{\bar{z}}}^-(\mathbb{X}, \mathcal{Z})$ .

*Remark 5.9.* An interesting question to study would be to consider if in this 2D case, every states of  $\text{Inv}_{\mathbf{F}_{\bar{z}}}^+(\mathbb{X})$  converge to  $\text{Viab}_{\mathbf{f}_{\bar{z}}}^-(\mathbb{X}, \mathcal{Z})$  under the assumption that  $\text{Viab}_{\mathbf{f}_{\bar{z}}}^-(\mathbb{X}, \mathcal{Z}) \subset \text{Inv}_{\mathbf{F}_{\bar{z}}}^+(\mathbb{X})$ .

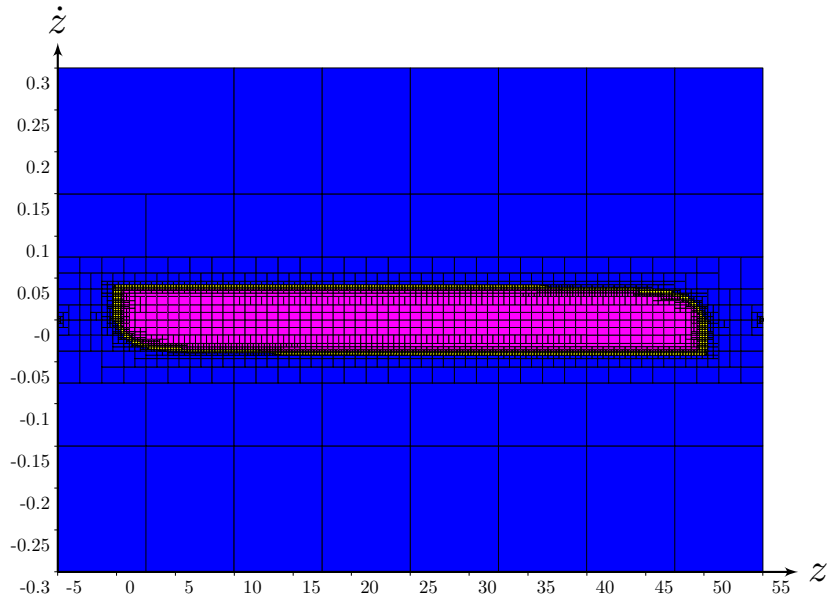


Figure 5.23: Largest viable set  $\text{Viab}_{f_z}^-(\mathbb{X}, \mathbb{Z})$ . Velocity ( $\text{m s}^{-1}$ ) in function of depth (m).

### 5.5.3 Additional validations

#### 5.5.3.1 Vector field following

An other important issue is to verify that the maximum piston volume rate  $\dot{V}_p$  is sufficient to follow the imposed trajectory, *i.e.*  $\forall t, u(t) \in [\dot{V}_p]$ .

We first consider that the float is stabilized on the trajectory (Le Gallic et al., 2018). We know that  $\dot{z} = \beta \arctan\left(\frac{\bar{z}-z}{\alpha}\right)$  and we also know that  $\dot{y} = 0$  so  $\ddot{z} = \gamma \frac{-\dot{z}}{D}$  which implies that  $V_p = \frac{1}{A} \left( \frac{\gamma \dot{z}}{D} + B |\dot{z}| \dot{z} \right) + \chi z$ . We can then compute  $u$  as a function of  $z$ , as we know  $\dot{z}$  and  $V_p$  in function of  $z$ . In order to obtain a guaranteed evaluation of  $u(z)$  and avoid numerical errors, we will use interval computations to bracket the value of the function and obtain two boundaries  $u^+$  and  $u^-$  such that  $u \in [u^-, u^+]$ .

If we take  $s = -1, \beta = 0.05 \frac{2}{\pi}, \alpha = 1$  and  $\bar{z} = 0$ , we obtain full black curves in Figure 5.24. We can see that the input is always inside the maximum and minimum piston volume rate. We can clearly see that when the system is far from the target, it needs to compensate  $\chi$ . The local increase of  $u$  near the set point is due to the deceleration. The maximum volume velocities computed are  $\{-1.383 \times 10^{-7}, 1.383 \times 10^{-7}\} \text{m}^3 \text{s}^{-1} \subset [\dot{V}_p]$ .

We can also study the behavior of the controller near the desired velocity set-point. If we assume now that  $\dot{z} = \beta \arctan\left(\frac{\bar{z}-z}{\alpha}\right) + w$  where  $w \in$

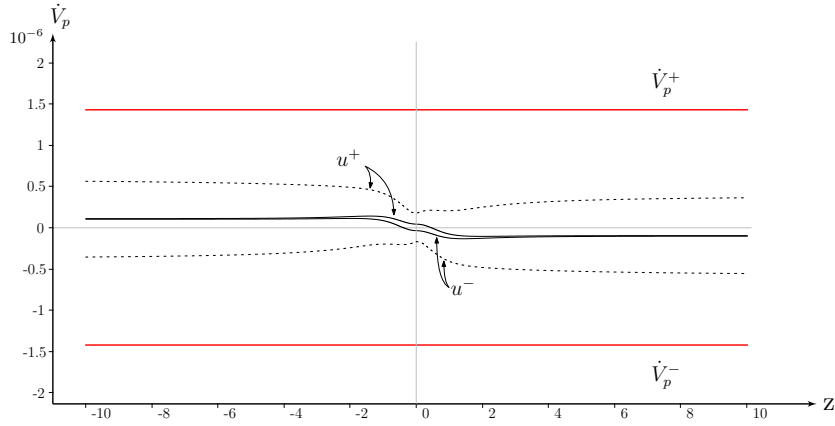


Figure 5.24: Outer approximation of the input in function of depth for two configurations. The maximum bound of  $\dot{V}_p$  are in red. Dotted curves correspond to the model with noise.

$[-5 \times 10^{-3}, 5 \times 10^{-3}] \text{ m s}^{-1}$ , we obtain the dotted curves of Figure 5.24 and the new maximum volume velocities:  $\{-5.62 \times 10^{-7}, 5.62 \times 10^{-7}\} \text{ m s}^{-1} \subset [\dot{V}_p]$ .

More generally, a study of the limitation of  $\dot{V}_p$  should be done for every states and not only in the case where the float is stabilized on the controller trajectory. This would however require at least to study the system in a 3D space.

### 5.5.3.2 Minimum piston volume increment

Defining the minimum codewheel step increment is a difficult task. A way to estimate a minimum value is to evaluate from Equation 5.9 which error of velocity produces a step of piston volume.

In the case of the *SeaBot*, if we allow a  $\delta \dot{z} = 0.01 \text{ m s}^{-1}$  velocity error at zero depth, we obtain an equivalent error of volume of  $\delta V_{p,\max} = 2.306 \times 10^{-7} \text{ m}^3$ . We have chosen, for the *SeaBot*, a 48 counts per revolution codewheel which gives a  $\delta V_p = 7.16 \times 10^{-8} \text{ m}^3 < \delta V_{p,\max}$ .

### 5.5.3.3 Energy consumption

For a given maximum velocity  $\dot{z}_m$  and a depth change  $\Delta z$  with a zero velocity at the beginning, and at the end, we can deduce, for the piston, the required variation of volume. By taking into account the piston volume rate, we can obtain the amount of motor run time if we suppose at a first approximation



a one-time movement. We can then deduce an under approximation of the power required for the mission:

$$\mathcal{E} = \frac{\Delta V_p}{\dot{V}_{p,\max}} \mathcal{P}_m = \frac{2 \left( \frac{B}{A} \dot{z}_m^2 \right) + |\chi \Delta z|}{\dot{V}_{p,\max}} \mathcal{P}_m$$

We assume here, to simplify, that the piston movement for the velocity and for the loss of volume are independent. We also assume that the piston has to first move to reach  $\dot{z}_m$  and then has to move to decelerate to zero velocity.

In the case of the *SeaBot*, for a  $\Delta z = 50$  m and  $\dot{z}_m = 0.05$  m s<sup>-1</sup>, we obtain a run time of 83s and  $\mathcal{E} = 0.457$  W h.

## 5.6 Design loop

Similarly to the ship design loop technique from the Naval Architecture community, we propose here a low-cost float design loop. The idea is to compute from the problem inputs, the minimum electronic and mechanical characteristics of the float.

- Problem inputs:
  - mass of the payload:  $m_p$ ,
  - maximum float velocity required  $\dot{z}_{\max}$ , trajectory to follow and error allowed,
  - volume of the antennas  $V_a$ ,
  - max depth:  $z_m$ ,
  - mission duration  $T$  and number of typical depth variation  $\Delta z$ .
- Design loop
  1. Compute the diameter  $d$  and length  $L$  of the float (function of  $m_p$ ). Choose a material for the hull and compute its minimum thickness (function of  $z_m$ ).
  2. Estimate the drag coefficient  $C_d$  (function of  $d$ ) and the loss of volume per meter  $\chi$ .
  3. Compute the interval volume of the piston  $[V_p]$  required (i) to emerge antennas ( $V_a$ ), (ii) to compensate the loss of volume ( $\chi$ ), (iii) and to reach the maximum velocity (function of  $C_d$ ).

4. Compute the interval velocity of the piston  $\left[\dot{V}_p\right]$  required to follow the trajectory and compensate  $\chi$ . Set the minimum motor specifications.
5. Compute the step increment of piston volume ( $\delta V$ ) and choose a depth sensor according to specifications.
6. Estimate the power consumption and choose the battery capacity.
7. Go to step 1 if energy autonomy does not comply with the maximum payload weight.

## 5.7 Conclusion and future work

In this chapter, we have proposed a mission strategy to answer the global problem of using ocean currents as the main source of propulsion. First of all, we looked at how the safety constraints on the path, which is assigned to a robot, could be translated into a problem involving *mazes*. We did not have sufficient time to develop the solution up to a real test case. Secondly, we have been dealing with the developments of a new kind of profiling float: a hybrid profiling float, that is able to modify its trajectory to use ocean currents. We have proposed a new depth controller, based on an EKF and a state feedback linearization, which was tested and validated using a hybrid float prototype. Finally, we have summarized our results with a design loop that could help future developments of profiling floats. This loop gives key parameters, that should be checked, for the design of a profiling float.

We have seen that *mazes* can be used at a mission level but also at a lower level with the validation, for instance, of a depth controller. We can note that the paths at the high level mission were not of the same nature as the ones at the low level. However, both problems can still be formalized using *mazes*.

There exist numerous perspectives to this chapter. First of all, we lack a validation of the whole system in a real environment. The limitation of the implementation of *mazes* to 2D systems did not allow to validate the complete controller in  $\mathbb{R}^3$ . Moreover, a validation of the coupling between the EKF and the controller should be studied to guarantee the convergence of the system.

# Chapter 6

## General conclusion and prospects

In this work we have been considering the problem of using oceanic currents as the main source of propulsion of underwater robots. To answer this challenge, we have proposed several new theoretical tools and also designed a new kind of underwater robot. Providing theoretical tools to guarantee that the robot will be able to carry out such missions has been a crosscutting concern of this work.

### 6.1 Summary of the Contributions

To answer the *Challenges* introduced in Chapter 1, we started by presenting in Chapter 2 the key concepts of dynamical systems and more specifically the notion of invariant sets. To handle these sets in a computer, we have introduced the framework of Abstract Interpretation and tools from Interval Analysis. We have, in addition, drawn some parallels between the two communities. These tools were interesting as they enable to guarantee formally the results of computations. We have also shown how a Constraint Programming approach along with the two previous fields could help to solve problems involving complex objects such as paths. Several examples have been given and in particular a contractor on dataset that allows to efficiently handle oceanic currents data.

To better handle paths and associated constraints, we have introduced in Chapter 3 a new abstract domain called *maze*. This chapter was focusing on bracketing the largest positive invariant sets of a dynamical system. We have shown how a set of new algorithms can be used efficiently to bracket these sets using *mazes*. We have also shown how to implement the algorithms using existing libraries. Moreover, we have presented some problems encountered with *mazes* such as the sliding problem or the limitation to low-dimensional

spaces. We have applied the tools to the bracket of the largest positive invariant set of the Van Der Pol system. Finally, we have shown how the theory could be extended to bracket the largest positive viable set.

In Chapter 4 we have shown how the bracket of invariant sets can be a basic tool to solve more complex problems. Several classic problems from the literature have been studied such as the bracket of forward and backward reach sets, attraction basins and capture reach sets. Finally, we have introduced a new Eulerian state estimator that formalizes in one framework problems involving paths. Indeed, we have shown how an Eulerian state estimator problem could be rewritten as the bracket of positive invariant sets. This new tool opens up perspectives on the design of a new language that could formalize problems involving paths.

Chapter 5 has focused on answering in practice the questions raised by the *Challenges*. We have proposed solutions at a high and low level. Concerning the high level, we have proposed a formalization of missions that should use oceanic currents. In particular, we have shown how guaranteeing that the robot will fulfill the mission, can be translated into an invariant sets problem. At a low level, we have introduced a new kind of hybrid profiling float to carry out such missions. This new kind of robot can regulate its depth with a new depth controller and can modify its trajectory using auxiliary thrusters. We have shown how the robot controller can be validated theoretically using *maze* tools. We have also validated the robot controller with experimental results. More generally, we have shown how tools such as *mazes* can help to prove the safety of robotic systems.

## 6.2 Prospects

We draw here some prospects concerning *maze* tools and oceanic current missions.

**Mazes** *Mazes* have appeared as a powerful tool to deal with dynamical systems in low dimensions. There exist an important number of prospects with their use. We will first consider ideas to improve already presented tools and in a second time an extension of the tools to solve new problem families.

First of all, a complete nD implementation which takes into account the sliding effect should be undertaken. More generally, the speed of convergence of *maze* algorithms could be improved by providing, among other, a better vector field approximation, a better bisection heuristic, a better constraint propagation heuristic, and a better implementation of contractors and inflators, . . . (see for more details Section 3.3.5). We have also seen that *mazes*

in dimension greater than two, rely on the use of convex polytopes. Nowadays, libraries that handle convex polytopes are based on rational numbers with exact computations. These libraries are relatively slow and require large memory space. A key work should be to implement a convex polytope floating-point library.

We now consider the extension of the algorithms to new problems. One of the most important perspectives is the possible extension of *maze* tools to solve hybrid dynamical systems problems (Goebel, Sanfelice, and Teel, 2009). Such systems are characterized by path jumps in the state space. These jumps occur when a guard condition is met. One of the ideas is to consider a directional virtual door of  $\mathbb{R}^n$  that could be added to each box of the paving to modeled jumps. This door would act as a wormhole in the maze. This corresponds to the addition of new edges in the maze graph. An other interesting perspective would be the study of the coupling between Eulerian and Lagrangian methods, such as *tubes* (Rohou et al., 2018). Each method allows different kind of constraints to be addressed. This would allow to increase the number of dynamical constraints that can be handled.

**Oceanic current missions** The second prospects sequence concerns oceanic current missions. As for the previous paragraph, we will propose short term and long term perspectives.

First of all, a complete experimental validation of the presented mission should be undertaken. Experiments always reveal unidentified problems. In particular, a focus should be given to an improvement of the energy consumption of the float, algorithmically and mechanically. For instance, the float piston must always counter a large pressure differential between the low pressure inside the float and the external high pressure due to depth. For most missions, the float need to stay at a constant depth and must apply small corrections to the piston position. Finding a way to reduce the pressure differential could be an interesting way to save energy. The float should also have a compressibility closed to the water compressibility to reduce the need to compensate for the difference. Concerning the depth control law, a better model that takes into account the temperature profile of the water could improve the efficiency of the stabilization. Indeed, if the float and water compressibility are close, the temperature profile is not anymore negligible.

Concerning the long term perspectives, the main issue with missions that use oceanic currents is the localization problem underwater. In this work, we have chosen indeed to return to the surface for path corrections. If we use a swarm of robots and if we measure the distance between the robots, we can obtain the measure of the distortion of the swarm under oceanic currents.

In some cases, this would allow to localize the swarm. Indeed, knowing the oceanic current map, there might exist only one evolution of the distortion that is compatible with the evolution of the position of the swarm.

Finally, oceanographers could be interested in using low cost hybrid profiling float. In the ocean there are interfaces between bodies of water that do not mix. These bodies of water do not have the same physical characteristics, particularly in terms of temperature. In these interfaces, internal waves are propagating. To study such waves, a swarm of robots could be deployed to follow an isotherm instead of an isobar. The variation of depth of the swarm would then allow the wave propagation to be reconstructed.

# Bibliography

- Agati, P., F. Lerouge, and M. Rossetto (2008). *Résistance des matériaux - 2ème édition - Cours, exercices et applications industrielles*. 2e édition. Paris: Dunod. 512 pp. ISBN: 978-2-10-051634-6.
- Asarin, Eugene, Thao Dang, and Antoine Girard (2003). “Reachability Analysis of Nonlinear Systems Using Conservative Approximation”. In: *Hybrid Systems: Computation and Control*. Ed. by Oded Maler and Amir Pnueli. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 20–35. ISBN: 978-3-540-36580-8.
- (2007). “Hybridization methods for the analysis of nonlinear systems”. In: *Acta Informatica* 43.7, pp. 451–476. ISSN: 1432-0525.
- Aubin, David and Amy Dahan Dalmedico (2002). “Writing the History of Dynamical Systems and Chaos: Longue Durée and Revolution, Disciplines and Cultures”. In: *Historia Mathematica* 29.3, pp. 273–339. ISSN: 0315-0860.
- Aubin, Jean-Pierre (2009). *Viability theory*. Reprint of the 1991 ed. Modern Birkhäuser classics. OCLC: 699336048. Boston, Mass.: Birkhäuser. 543 pp. ISBN: 978-0-8176-4910-4 978-0-8176-4909-8.
- Aublin, M. et al. (2005). *Systèmes mécaniques - Théorie et dimensionnement*. Paris: Dunod. 688 pp. ISBN: 978-2-10-049104-9.
- Bagnara, Roberto, Patricia M. Hill, and Enea Zaffanella (2008). “The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems”. In: *Science of Computer Programming*. Special Issue on Second issue of experimental software and toolkits (EST) 72.1, pp. 3–21. ISSN: 0167-6423.
- Barreiro, A., J. Aracil, and D. Pagano (2002). “Detection of attraction domains of non-linear systems using bifurcation analysis and Lyapunov functions”. In: *International Journal of Control* 75.5, pp. 314–327. ISSN: 0020-7179.
- Bayen, Alexandre M. et al. (2007). “Aircraft Autolander Safety Analysis Through Optimal Control-Based Reach Set Computation”. In: *Journal of Guidance, Control, and Dynamics* 30.1, pp. 68–77.

- Berkenpas, E. J. et al. (2018). “A Buoyancy-Controlled Lagrangian Camera Platform for In Situ Imaging of Marine Organisms in Midwater Scattering Layers”. In: *IEEE Journal of Oceanic Engineering* 43.3, pp. 595–607. ISSN: 0364-9059.
- Bessa, W. M. et al. (2017). “Design and Adaptive Depth Control of a Micro Diving Agent”. In: *IEEE Robotics and Automation Letters* 2.4, pp. 1871–1877. ISSN: 2377-3766.
- Biemond, J. J. Benjamin and Wim Michiels (2014). “Estimation of basins of attraction for controlled systems with input saturation and time-delays”. In: *IFAC Proceedings Volumes*. 19th IFAC World Congress 47.3, pp. 11006–11011. ISSN: 1474-6670.
- Blanchini, Franco and Stefano Miani (2015). *Set-theoretic methods in control*. OCLC: 946766307. ISBN: 978-3-319-17933-9.
- Bouissou, O. et al. (2014). “Computation of parametric barrier functions for dynamical systems using interval analysis”. In: *53rd IEEE Conference on Decision and Control*. 53rd IEEE Conference on Decision and Control, pp. 753–758.
- Chabert, G and L. Jaulin (2009). “Contractor programming”. In: *Artificial Intelligence* 173.11, pp. 1079–1100. ISSN: 00043702.
- Combastel, C. (2005). “A State Bounding Observer for Uncertain Non-linear Continuous-time Systems based on Zonotopes”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. Proceedings of the 44th IEEE Conference on Decision and Control, pp. 7228–7234.
- Cousot, Patrick (2001). “Abstract Interpretation Based Formal Methods and Future Challenges”. In: *Informatics: 10 Years Back, 10 Years Ahead*. Ed. by Reinhard Wilhelm. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 138–156. ISBN: 978-3-540-44577-7.
- Cousot, Patrick and Radhia Cousot (1976). “Static determination of dynamic properties of programs”. In: *Proceedings of the 2nd International Symposium on Programming, Paris, France*, pp. 106–130.
- (1977). “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '77*. the 4th ACM SIGACT-SIGPLAN symposium. Los Angeles, California: ACM Press, pp. 238–252.
- (1979). “Systematic design of program analysis frameworks”. In: *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '79*. the 6th ACM SIGACT-SIGPLAN symposium. San Antonio, Texas: ACM Press, pp. 269–282.



- (1992a). “Abstract Interpretation Frameworks”. In: *Journal of Logic and Computation* 2.4, pp. 511–547.
- (1992b). “Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation”. In: *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*. PLILP '92. Berlin, Heidelberg: Springer-Verlag, pp. 269–295. ISBN: 978-3-540-55844-6.
- Cousot, Patrick and Nicolas Halbwachs (1978). “Automatic Discovery of Linear Restraints Among Variables of a Program”. In: *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '78. event-place: Tucson, Arizona. New York, NY, USA: ACM, pp. 84–96.
- Cousot, Patrick et al. (2005). “The ASTREÉ Analyzer”. In: *Programming Languages and Systems*. Ed. by Mooly Sagiv. Red. by David Hutchison et al. Vol. 3444. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 21–30. ISBN: 978-3-540-25435-5 978-3-540-31987-0.
- Creuze, Vincent (2014). “Robots marins et sous-marins Perception, modélisation, commande”. In: *Techniques de l'ingénieur Applications en robotique base documentaire : TIB623DUO*. (ref. article : s7783).
- Daney, David, Yves Papegay, and Arnold Neumaier (2004). “Interval Methods for Certification of the Kinematic Calibration of Parallel Robots”. In:
- Davey, B. A. and H. A. Priestley (2002). *Introduction to lattices and order*. 2nd ed. Cambridge, UK ; New York, NY: Cambridge University Press. 298 pp. ISBN: 978-0-521-78451-1.
- Delanoue, Nicolas, Luc Jaulin, and Bertrand Cottenceau (2006). “Attraction domain of a non-linear system using interval analysis”. In: *International Conference on Principles and Practice of Constraint Programming (CP'06)*. Nantes, France.
- Desilles, Anna, Hasnaa Zidani, and Eva Crück (2012). “Collision analysis for an UAV”. In: *AIAA GUIDANCE, NAVIGATION, AND CONTROL CONFERENCE*. Minneapolis, United States, AIAA 2012–4526.
- Drevelle, V. and P. Bonnifait (2013). “Localization Confidence Domains via Set Inversion on Short-Term Trajectory”. In: *IEEE Transactions on Robotics* 29.5, pp. 1244–1256. ISSN: 1552-3098.
- Dubins, L. E. (1957). “On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents”. In: *American Journal of Mathematics* 79.3, pp. 497–516. ISSN: 0002-9327.

- Esterhuizen, Willem and Jean Lévine (2016). “Barriers and potentially safe sets in hybrid systems: Pendulum with non-rigid cable”. In: *Automatica* 73, pp. 248–255. ISSN: 0005-1098.
- Fossen, Thor I. (2011). *Handbook of marine craft hydrodynamics and motion control =: Vademecum de navium motu contra aquas et de motu gubernando*. OCLC: ocn699374017. Chichester, West Sussex: Wiley. 575 pp. ISBN: 978-1-119-99149-6.
- Frehse, Goran et al. (2011). “SpaceEx: Scalable Verification of Hybrid Systems”. In: *Computer Aided Verification*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 379–395. ISBN: 978-3-642-22109-5 978-3-642-22110-1.
- Gao, Yan, John Lygeros, and Marc Quincampoix (2006). “The Reachability Problem for Uncertain Hybrid Systems Revisited: A Viability Theory Perspective”. In: *Hybrid Systems: Computation and Control*. Ed. by João P. Hespanha and Ashish Tiwari. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 242–256. ISBN: 978-3-540-33171-1.
- Girard, Antoine, Colas Le Guernic, and Oded Maler (2006). “Efficient Computation of Reachable Sets of Linear Time-Invariant Systems with Inputs”. In: *Hybrid Systems: Computation and Control*. Ed. by João P. Hespanha and Ashish Tiwari. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 257–271. ISBN: 978-3-540-33171-1.
- Goebel, Rafal, Ricardo G. Sanfelice, and Andrew R. Teel (2009). “Hybrid dynamical systems”. In: *IEEE Control Systems* 29.2, pp. 28–93. ISSN: 1066-033X, 1941-000X.
- Gonzaga, Carlos A. C., Marc Jungers, and Jamal Daafouz (2012). “Stability analysis and stabilisation of switched nonlinear systems”. In: *International Journal of Control* 85.7, pp. 822–829. ISSN: 0020-7179.
- Goualard, Frederic. “Gaol: NOT Just Another Interval Arithmetic Library”. In: p. 92.
- Goubault, Eric and Sylvie Putot (2017). “Forward Inner-Approximated Reachability of Non-Linear Continuous Systems”. In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. HSCC '17. event-place: Pittsburgh, Pennsylvania, USA. New York, NY, USA: ACM, pp. 1–10. ISBN: 978-1-4503-4590-3.
- Gouttefarde, Marc, David Daney, and Jean-Pierre Merlet (2011). “Interval-Analysis-Based Determination of the Wrench-Feasible Workspace of Parallel Cable-Driven Robots”. In: *IEEE Transactions on Robotics* 27.1, pp. 1–13. ISSN: 1552-3098, 1941-0468.
- Guyonneau, R., S. Lagrange, and L. Hardouin (2013). “A visibility information for multi-robot localization”. In: *2013 IEEE/RSJ International*

- Conference on Intelligent Robots and Systems*. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1426–1431.
- Halbwachs, Nicolas, Yann-Erick Proy, and Patrick Roumanoff (1997). “Verification of Real-Time Systems using Linear Relation Analysis”. In: *Formal Methods in System Design* 11.2, pp. 157–185. ISSN: 1572-8102.
- Huth, Michael and Mark Ryan (2004). *Logic in computer science: modelling and reasoning about systems*. 2nd ed. Cambridge [U.K.] ; New York: Cambridge University Press. 427 pp. ISBN: 978-0-521-54310-1.
- “IEEE Standard for Interval Arithmetic” (2015). In: *IEEE Std 1788-2015*, pp. 1–97.
- Jaffe, J. S. et al. (2017). “A swarm of autonomous miniature underwater robot drifters for exploring submesoscale ocean dynamics”. In: *Nature Communications* 8, p. 14189. ISSN: 2041-1723.
- Jaulin, L. (2006). “Computing minimal-volume credible sets using interval analysis; application to bayesian estimation”. In: *IEEE Transactions on Signal Processing* 54.9, pp. 3632–3636. ISSN: 1053-587X.
- (2015). *Mobile Robotics*. Ed. by ISTE. Mobile Robotics.
- (2018). “Isobath Following using an Altimeter as a Unique Exteroceptive Sensor”. In: *Robotic Sailing 2018*. Ed. by Sophia M. Schillai and Nicholas Townsend. event-place: Southampton, UK, pp. 105–110.
- Jaulin, L. and B. Desrochers (2014). “Introduction to the algebra of separators with application to path planning”. In: *Engineering Applications of Artificial Intelligence* 33, pp. 141–147. ISSN: 0952-1976.
- Jaulin, L. et al. (2001). *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. OCLC: 853266316. London: Springer London. ISBN: 978-1-4471-0249-6.
- Kalies, William D., Konstantin Mischaikow, and Robert C. A. M. Vanderhorst (2016). “Lattice Structures for Attractors II”. In: *Foundations of Computational Mathematics* 16.5, pp. 1151–1191. ISSN: 1615-3383.
- Kaynama, Shahab et al. (2012). “Computing the viability kernel using maximal reachable sets”. In: *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control - HSCC '12*. the 15th ACM international conference. Beijing, China: ACM Press, p. 55. ISBN: 978-1-4503-1220-2.
- Khalil, Hassan K. (2002). *Nonlinear systems*. 3rd ed. Upper Saddle River, N.J: Prentice Hall. 750 pp. ISBN: 978-0-13-067389-3.
- LaSalle, Joseph P and Solomon Lefschetz (1961). *Stability by Liapunov’s direct method with applications*. OCLC: 428097379. Amsterdam: Elsevier Science. ISBN: 978-0-12-437056-2.

- Lazure, Pascal and Franck Dumas (2008). “An external–internal mode coupling for a 3D hydrodynamical model for applications at regional scale (MARS)”. In: *Advances in Water Resources* 31.2, pp. 233–250. ISSN: 0309-1708.
- Le Gallic, M. et al. (2018). “Tight slalom control for sailboat robots”. In: International Robotic Sailing Conference (IRSC).
- Le Mézo, Thomas, Luc Jaulin, and Benoit Zerr (2017a). “An Interval Approach to Compute Invariant Sets”. In: *IEEE Transactions on Automatic Control* 62.8, pp. 4236–4242. ISSN: 0018-9286.
- (2017b). “Eulerian state estimation”. In: SWIM’17.
- (2018). “Bracketing the solutions of an ordinary differential equation with uncertain initial conditions”. In: *Applied Mathematics and Computation. Recent Trends in Numerical Computations: Theory and Algorithms* 318, pp. 70–79. ISSN: 0096-3003.
- (2019). “Bracketing backward reach sets of a dynamical system”. In: *International Journal of Control*, pp. 1–13. ISSN: 0020-7179.
- Le Mézo, Thomas et al. (2019). “Design and control of a low-cost autonomous profiling float”. In: *24ème Congrès Français de Mécanique*. Brest, France.
- Le Reste, S. et al. (2016). “"Deep-Arvor": A new profiling float to extend the Argo observations down to 4000m depth.” In: *Journal Of Atmospheric And Oceanic Technology* 33.5, pp. 1039–1055. ISSN: 0739-0572.
- Lhommeau, M., L. Jaulin, and L. Hardouin (2011). “Capture basin approximation using interval analysis”. In: *International Journal of Adaptive Control and Signal Processing* 25.3, pp. 264–272. ISSN: 1099-1115.
- MahmoudZadeh, S. et al. (2018). “Online path planning for AUV rendezvous in dynamic cluttered undersea environment using evolutionary algorithms”. In: *Applied Soft Computing* 70, pp. 929–945. ISSN: 1568-4946.
- McGilvray, B. and C. Roman (2010). “Control system performance and efficiency for a mid-depth Lagrangian profiling float”. In: *OCEANS’10 IEEE SYDNEY*. OCEANS’10 IEEE SYDNEY, pp. 1–10.
- Miné, Antoine (2006). “The Octagon Abstract Domain”. In: *Higher Order Symbol. Comput.* 19.1, pp. 31–100. ISSN: 1388-3690.
- (2013). “Static analysis by abstract interpretation of concurrent programs”. thesis.
- Mitchell, Ian M. (2007). “Comparing Forward and Backward Reachability as Tools for Safety Analysis”. In: *Hybrid Systems: Computation and Control*. Ed. by Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 428–443. ISBN: 978-3-540-71493-4.

- Mitchell, Ian, Alexandre M. Bayen, and Claire J. Tomlin (2001). “Validating a Hamilton-Jacobi Approximation to Hybrid System Reachable Sets”. In: *Hybrid Systems: Computation and Control*. Ed. by Maria Domenica Di Benedetto and Alberto Sangiovanni-Vincentelli. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 418–432. ISBN: 978-3-540-45351-2.
- Montanari, Ugo (1974). “Networks of constraints: Fundamental properties and applications to picture processing”. In: *Information Sciences* 7, pp. 95–132. ISSN: 0020-0255.
- Moore, R.E. (1966). *Interval analysis*. Prentice-Hall series in automatic computation. Prentice-Hall.
- Pelleau, Marie et al. (2013). “A Constraint Solver Based on Abstract Domains”. In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 434–454. ISBN: 978-3-642-35873-9.
- Pineau-Guillou, Lucia (2013). *PREVIMER. Validation des modèles hydrodynamiques 2D des côtes de la Manche et de l’Atlantique*.
- Quincampoix, M. (1992). “Differential Inclusions and Target Problems”. In: *SIAM Journal on Control and Optimization* 30.2, pp. 324–335. ISSN: 0363-0129.
- Rakovic, S. V. et al. (2005). “Invariant approximations of the minimal robust positively invariant set”. In: *IEEE Transactions on Automatic Control* 50.3, pp. 406–410.
- Rao, Dushyant and Stefan B Williams (2009). “Large-scale path planning for Underwater Gliders in ocean currents”. In: p. 8.
- Ratschan, Stefan and Zhikun She (2010). “Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-Like Functions”. In: *SIAM Journal on Control and Optimization* 48.7, pp. 4377–4394. ISSN: 0363-0129, 1095-7138.
- Riser, S. C. et al. (2016). “Fifteen years of ocean observations with the global Argo array”. In: *Nature Climate Change* 6.2, pp. 145–153. ISSN: 1758-6798.
- Rohou, Simon et al. (2017). “Guaranteed computation of robot trajectories”. In: *Robotics and Autonomous Systems* 93, pp. 76–84. ISSN: 0921-8890.
- (2018). “Reliable non-linear state estimation involving time uncertainties”. In: *Automatica* 93, pp. 379–388. ISSN: 0005-1098.
- Rohou, Simon et al. (2019). *Reliable robot localization: a constraint-programming approach over dynamical systems*. Hoboken: ISTE Ltd / John Wiley and Sons Inc. ISBN: 978-1-84821-970-0.

- Rossi, Francesca, Peter Van Beek, and Toby Walsh, eds. (2006). *Handbook of constraint programming*. 1st ed. Foundations of artificial intelligence. OCLC: ocm70408044. Amsterdam ; Boston: Elsevier. 955 pp. ISBN: 978-0-444-52726-4.
- Russell, Stuart J., Peter Norvig, and Ernest Davis (2010). *Artificial intelligence: a modern approach*. 3rd ed. Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall. 1132 pp. ISBN: 978-0-13-604259-4.
- Saint-Pierre, Patrick (1994). "Approximation of the viability kernel". In: *Applied Mathematics and Optimization* 29.2, pp. 187–209. ISSN: 1432-0606.
- (2002). "Hybrid Kernels and Capture Basins for Impulse Constrained Systems". In: *Hybrid Systems: Computation and Control*. Ed. by Claire J. Tomlin and Mark R. Greenstreet. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 378–392. ISBN: 978-3-540-45873-9.
- Sandretto, Julien Alexandre dit and Alexandre Chapoutot (2016). "Validated Simulation of Differential Algebraic Equations with Runge-Kutta Methods". In: *Reliable Computing electronic edition*. Special issue devoted to material presented at SWIM 2015 22.
- Schiaretti, Matteo, Linying Chen, and Rudy R. Negenborn (2017). "Survey on Autonomous Surface Vessels: Part II - Categorization of 60 Prototypes and Future Applications". In: *Computational Logistics*. Ed. by Tolga Bektaş et al. Lecture Notes in Computer Science. Springer International Publishing, pp. 234–252. ISBN: 978-3-319-68496-3.
- Seube, N., R. Moitie, and G. Leitmann (2000). "Aircraft Take-Off in Wind-shear: A Viability Approach". In: *Set-Valued Analysis* 8.1, pp. 163–180. ISSN: 1572-932X.
- Shi, Y. et al. (2017). "Advanced Control in Marine Mechatronic Systems: A Survey". In: *IEEE/ASME Transactions on Mechatronics* 22.3, pp. 1121–1131. ISSN: 1083-4435.
- Slotine, J.-J. E. and Weiping Li (1991). *Applied nonlinear control*. Englewood Cliffs, N.J: Prentice Hall. 459 pp. ISBN: 978-0-13-040890-7.
- Smith, Ryan N. et al. (2010). "Planning and Implementing Trajectories for Autonomous Underwater Vehicles to Track Evolving Ocean Processes Based on Predictions from a Regional Ocean Model". In: *The International Journal of Robotics Research* 29.12, pp. 1475–1497.
- Taha, Walid et al. (2016). "Acumen: An Open-Source Testbed for Cyber-Physical Systems Research". In: *Internet of Things. IoT Infrastructures*. Ed. by Benny Mandler et al. Vol. 169. Cham: Springer International Publishing, pp. 118–130. ISBN: 978-3-319-47062-7 978-3-319-47063-4.

- Tahir, Furqan and Imad M. Jaimoukha (2012). “Robust Positively Invariant Sets for Linear Systems subject to model-uncertainty and disturbances”. In: *IFAC Proceedings Volumes*. 4th IFAC Conference on Nonlinear Model Predictive Control 45.17, pp. 213–217. ISSN: 1474-6670.
- Tarski, Alfred (1955). “A lattice-theoretical fixpoint theorem and its applications.” In: *Pacific Journal of Mathematics* 5.2, pp. 285–309. ISSN: 0030-8730.
- Timoshenko, S. (1941). “Strength of Materials, Part II”. In: *Advanced Theory and Problems* 245.
- Wilczak, D. and P. Zgliczyński (2011). “Cr - Lohner algorithm”. In: *Schedae Informaticae* Vol. 20, pp. 9–42.
- Zereik, Enrica et al. (2018). “Challenges and future trends in marine robotics”. In: *Annual Reviews in Control* 46, pp. 350–368. ISSN: 1367-5788.

**Titre :** Encadrement des plus grands ensembles invariants de systèmes dynamiques : une application aux robots sous-marins dérivant dans les courants océaniques.

**Mot clés :** robotique sous-marine, systèmes dynamiques, ensembles invariants, interprétation abstraite, programmation par contraintes, flotteurs profileurs.

**Résumé :** La vérification de la sûreté de fonctionnement des systèmes robotiques est une question fondamentale pour le développement de la robotique. Elle consiste, par exemple, à vérifier qu'une loi de commande d'un robot respectera toujours un ensemble de contraintes. Plus généralement, nous nous intéresserons ici à la vérification des propriétés de systèmes dynamiques, ces derniers permettant de modéliser l'évolution d'un robot.

La contribution principale de cette thèse est d'apporter un nouveau moyen d'encadrer les ensembles invariants de systèmes dynamiques. Pour cela, un nouveau domaine abstrait, les mazes, et de nouveaux algorithmes sont présentés. Il est également montré, au travers de nombreux exemples, comment des problèmes classiques de validation peuvent être ramenés à un problème d'encadrement d'ensembles invariants. Enfin, les résultats

sont étendus à l'encadrement des noyaux de viabilité.

Cette thèse s'appuie également sur une application en robotique sous-marine. L'idée principale est d'utiliser les courants marins pour qu'un robot sous-marin puisse parcourir avec efficacité de grandes distances. Un nouveau type de robot autonome bas coût a été développé pour ce type de mission. Ce nouveau flotteur profileur hybride est capable de se réguler en profondeur grâce à une nouvelle loi de régulation, mais également de corriger sa trajectoire à l'aide de propulseurs auxiliaires. Ils permettent au robot de choisir la bonne veine de courant à emprunter. Les outils de validations précédemment introduits sont utilisés pour valider la sûreté du robot et de la mission. Des expérimentations en conditions réelles ont également permis de valider le prototype.

**Title:** Bracketing largest invariant sets of dynamical systems: an application to drifting underwater robots in ocean currents.

**Keywords:** underwater robotics, dynamical systems, invariant sets, abstract interpretation, constraint programming, profiling floats.

**Abstract:** The proof of safety of robotic systems is a fundamental issue for the development of robotics. It consists, for instance, in verifying that a robot control law will always satisfy a set of constraints. More generally, we will be interested here in the verification of the properties of dynamical systems, as the latter allow to model the evolution of a robot.

The main contribution of this thesis is to provide a new way of bracketing invariant sets of dynamical systems. To this end, a new abstract domain, the mazes, and new algorithms are presented. It is also shown, through many examples, how classic validation problems can be translated into a problem of bracketing invariant sets. Finally, the results are ex-

tended to the bracket of viability kernels.

This thesis is also based on an application in underwater robotics. The main idea is to use ocean currents so that an underwater robot can efficiently travel long distances. A new kind of low-cost autonomous robot has been developed for this type of mission. This new hybrid profiling float is able to regulate its depth with a new regulation law, but also to correct its trajectory using auxiliary thrusters. They allow the robot to choose the right flow of current to be used. The previously introduced validation tools are applied to validate the robot and the mission safety. Experiments in real conditions also enabled the prototype to be validated.