Lie groups applied to localisation of mobile robots

Julien DAMERS

April 2022, 25th

CONTENTS

ABSTRACT

Keywords
Dynamical systems, Guaranteed integration, Lie symmetries, Localisation

With the development of offshore activities the costs of maintenance and monitoring of offshore plants in terms of crew members, boats, and money has greatly increased and is still growing dramatically. This encouraged the development of Autonomous Underwater Vehicles (AUV). These are still very expensive because of the numerous high-end sensors they need to embark to accomplish their missions. Thus their number is relatively low. Therefore research is made to develop low-cost AUVs that could be produced in a larger amount to perform the same missions. This thesis comes within the scope of this research field. One of the main problems when dealing with AUVs is the localisation of the vehicle which will be the one addressed throughout this work. To tackle it, we present a new guaranteed integration method, which is more robust to the uncertainties on the initial condition than the ones currently available, based on Lie symmetries. This method is first presented through different simple theoretical examples. We then apply it on a the localisation problem in a robotic context.

RÉSUMÉ

Mots clés
Systèmes dynamiques, Intégration garantie, Symétries de Lie, Localisation

Dans le cadre du développement des énergies renouvelables, les activités offshores liées aux parcs éoliens ont grandement augmentés en nombre. Ainsi les coûts de maintenance et de surveillance de telles structures aussi bien en hommes qu'en moyens se sont accrus et cette tendance s'amplifiera sûrement au cours des prochaines années. Ceci a encouragé le développement des véhicules sous-marins autonomes (AUV). Ceux-ci sont encore très coûteux du fait des capteurs onéreux qu'ils embarquent. Ils sont donc encore rare, ce qui ne suffit pas à combler les actuels besoins. Ainsi le développement d'AUVs à bas-coûts est un sujet de recherche en fort croissance. C´est dans ce champ de recherche que cette thèse vient s'inscrire. Un des problèmes principaux rencontré en robotique sous-marine est la localisation de l'engin. C'est la résolution celui-ci qui nous servira d'objectif tout au long de ce travail. Pour ce faire, une nouvelle méthode d'intégration garantie, robuste aux incertitudes sur les conditions initiales est présentées dans ce manuscrit. Celle-ci se base sur l'utilisation des symétries de Lie appliquées aux équations différentielles. Nous la présenterons en premier sur des problèmes théoriques simples avant de l'appliquer sur le problème de localisation d'un robot sous-marin.

ACRONYMS

AHRS Attitude and Heading Reference System

AUV Autonomous Underwater Vehicle

CAPD Computer Assisted Proof in Dynamic groups

CN Constraint Network

Codac Catalog Of Domains And Contractors

CSP Constraint Satisfaction Problem

CtcCN Contractor on the Contractor Network

DVL Doppler Velocity Log

FOE First Order Enclosure

GNSS Global Navigation Satellite System

GPS Global Positioning System

IBEX Interval Based Explorer

INS Inertial Navigation System

IVP Initial Value Problem

LBL Long Baseline

ODE Ordinary Differential Equation

PF Particle Filter

ROV Remotely Operated Vehicle

SIVIA Set Inversion *via* Interval Analysis

TC1 Test-Case 1

TC2 Test-Case 2

TC3 Test-Case 3

TC4 Test-Case 4

USBL Ultra Short Baseline

NOTATIONS

General notations

x, α, T	Scalar values
$\mathbf{x} = (x, y, z) = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	Vector value
$f(\dots)$	Scalar-valued function
$\mathbf{f}(\dots)$	Vector-valued function

Interval Analysis

$[x]$	Interval of $\mathbb{R}$
$[\mathbf{x}]$	Interval vector (box) of $\mathbb{R}^n$
$\mathbf{f}([\mathbf{x}])$	Image of $[\mathbf{x}]$ by $\mathbf{f}$
$[\mathbf{f}]([\mathbf{x}])$	Inclusion function for $\mathbf{f}([\mathbf{x}])$
$[\mathbf{f}]^*([\mathbf{x}])$	Minimal inclusion function for $\mathbf{f}([\mathbf{x}])$
$[\mathbf{f_c}]^*([\mathbf{x}])$	Centred form inclusion function for $\mathbf{f}([\mathbf{x}])$
c_i	Constraint associated with the function f_i
$\mathcal{C}_i$	Contractor associated with the constraint c_i
$\mathcal{S}_i$	Separator associated with the constraint c_i

Tubes

$x(\cdot)$	One dimensional trajectory
$\mathbf{x}(\cdot)$	n-dimensional trajectory
$[x](\cdot)$	One dimensional tube
$[\mathbf{x}](\cdot)$	n-dimensional tube
$[\mathbf{x}](t)$	Evaluation of the tube $[\mathbf{x}]$ at time t ($t \in \mathbb{R}$)

Sets

$\mathbb{R}$	Set of real numbers
$\mathbb{IR}$	Set of intervals of $\mathbb{R}$
$\mathbb{IR}^n$	Set of axis-aligned boxes of $\mathbb{R}$
$\mathcal{C}^m(\mathbb{R}^n)$	Set of the functions of $\mathbb{R}^n$ with m continuous derivatives
$[\mathbf{x}](t)$	Evaluation of the tube $[\mathbf{x}]$ at time t ($t \in \mathbb{R}$)
$\emptyset$	Empty set

Lie integration method

$a, \mathbf{a}, a(\cdot), [a](\cdot), \mathbf{a}(\cdot), [\mathbf{a}](\cdot)$	Reference
$\mathbf{g_p}$	Symmetry function
$\mathbf{h}(\mathbf{x}, \mathbf{a})$	Transport function from $\mathbf{x}$ to $\mathbf{a}$
$\mathbb{X}_t$	Solution set at time t
$\mathbf{\Phi}_t(\mathbf{x})$	Flow function

CHAPTER 1

INTRODUCTION

Contents

1.1 Context

1.1.1 An unknown environment

Oceans remain rather unknown to humankind despite they cover over seventy percent of our planet. According to the National Oceanic and Atmospheric Administration (NOAA), more than 80% of it are yet to be explored. One may even read that with the previous moon observations and current Mars exploring missions, we got more acquainted with these two extraterrestrial territories during the last 70 years than with the oceans of our own mother planet. However, the study of the maritime environment is nothing new. In the late 19^{th} century, expeditions were already carried out to measure the depth in different areas using probes attached to a surface vessel. It is during one of these operations that the deepest point one Earth, Challenger Deep, was discovered. It was named after the expedition and vessel name that performed the survey, the HMS Challenger. However if the first idea of what is nowadays called submarines can be found in the Antiquity (mainly for military purposes), humankind started using them for deep-sea exploration in the beginning of the 1930s with the *Bathysphere*. Since then an astonishing amount of progress has been made in the field leading to the development of nuclear submarines, considered as one of the most advanced human achievement.

Figure 1.1: The original *Bathysphere* that was used by William Beebe and Otis Barton to set the first record of deep sea dive at 245m in 1930 [44].

1.1.2 We still need to work with

With the ever growing need for energy and more communications between the different continents we started the exploitation of this unknown environment.

Concerning energy production, we first encouraged the development of offshore plants to collect petroleum and fossil resources. However, with the dramatic increase of the demand, the quantity of this non-renewable energy extracted now exceeds by far the quantity slowly produced. Leading to non avoidable future shortages. Moreover, the use of such energy is at the expense of great environmental costs by affecting the atmosphere, with greenhouse gases, the fauna and flora. Thus our civilisation is progressively turning to renewable sources of energy. The main sources of renewable energy are of the count of three, namely hydropower, solar power an wind (both onshore and offshore). In the last decade, the share of renewable energy produced has been steadily increasing in OECD Europe according to the International Energy Agency. Among them, the coastal and offshore power are subject to a lot of investments with the development of wind farms or hydro turbines. If the latter is lesser known, the use of tidal streams is heavily studied in order to benefit from the enormous source of power the ocean can provide.

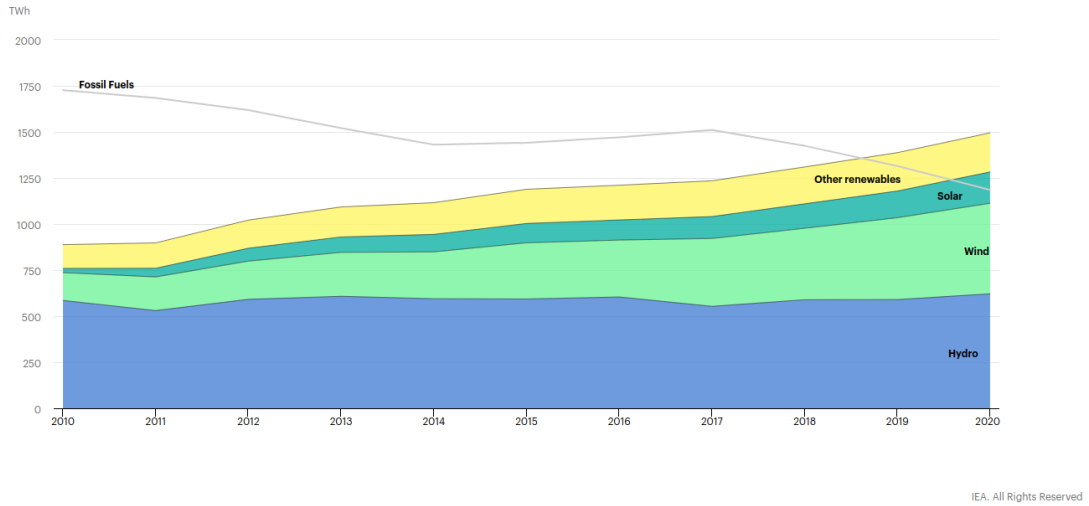


Figure 1.2: Evolution of energy production from different sources in OECD from 2010 to 2020 [56].

On the communication aspect, continents have been linked together with a vast network of cables lying on the seabed for quite a long time now. It first started with telegraphs cables, but this network now enables the tremendous exchange of data between continents through the Internet. They are of vital importance in an era where countries are perpetually interconnected, from a military, economic or social point of views.

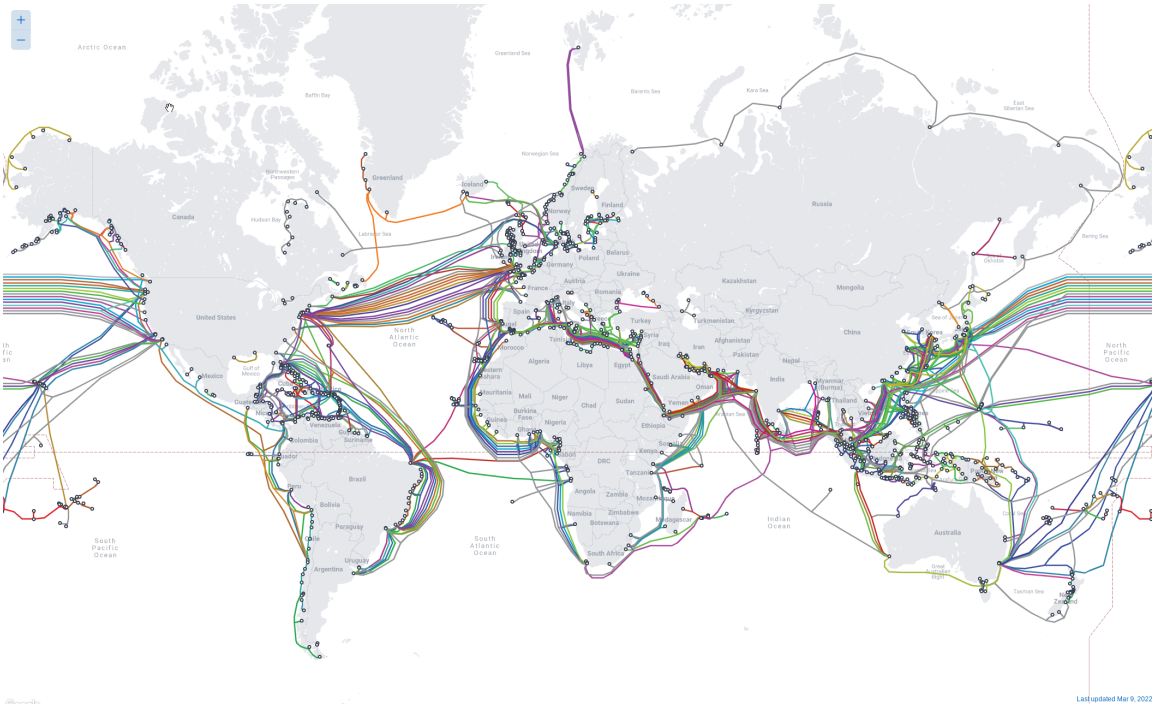


Figure 1.3: Map of the submarine cables in March 2022 [114].

All these offshore infrastructures require more maintenance and inspection than their onshore counterparts. Indeed, they suffer from additional sources of threat to their integrity such as corrosion, pressure or simply marine life. For a long time, highly skilled divers were in charge of these operations alone. They are now assisted by Remotely Operated Vehicle (ROV)s, which are robots deployed from the plant itself or from a surface vessel linked with a tether to the surface and remotely piloted by a trained technician. However, this is not ideal in terms of safety because of the risks of deep diving, especially in an industrial environment with multiple potential sources of accident. In addition, the need for a surface vessel, and as a consequence more technicians to operate, increases the cost of such operations. Thus, the monitoring of such structures is not permanent, multiplying the risks of undetected and potentially critical failures. With offshore plants flourishing around the globe, the need for a cheaper and safer monitoring and maintenance solution came quickly.

1.2 Towards the development of low costs autonomous underwater robots: a great challenge

1.2.1 Getting rid of the tether

As mentioned in Section 1.1.2, maintenance operations are now handled with the help of ROVs. These robots linked to the surface with a tether providing both power and communication to the surface are of great help when it comes to work in restrained area. Indeed the size of the zone of operations is limited by the length of the cable it is attached

to. If this limitation is not a real problem when dealing with an oil and gas offshore plant as the surface it covers is relatively limited, it becomes a critical issue when dealing with wind farms that cover a few square kilometres or monitoring thousand kilometres long cables. To perform the work efficiently, one would need many vessels to cover the area which means numerous crew members and hence a dramatic increase of safety risks and costs. Therefore the need for a wireless solution.

1.2.2 Why autonomous?

A wireless solution does not mean that it should be autonomous. In the same way it is possible to control an aerial drone with a joystick, one may envision some kind of radio signal to control the robot from distance. Unfortunately, radio signals do not travel through water, therefore we need another way to communicate with our vehicle.

Acoustic waves propagate very well underwater. They can cover long distances up to hundreds of kilometres depending on their wavelength. The higher the latter is, the greater the distance. However, a higher wavelength means a lower frequency. Which leads us to the main drawback of this technology which is the particularly low data throughput. For a range of a hundred metres, the data rate is around 5 Kbps [67] which is far from enough to operate a robot especially when doing complex manoeuvres where a live visual feed is essential for the operator.

Another solution might be optical communications. If acoustic communications underwater have been studied for decades, this field is relatively new and is currently the subject of a lot of research. It emerged in the late 90s but has been significantly pushed during the last decade. This is due to the development of underwater sensor fields which are often deployed in strategic areas, such as energy-related offshore plants, to monitor them. As they gather heavy loads of data, there is a need for a fast mean of communications underwater. It has been proven that this technology outperforms acoustic communication by far in terms of data rate with an average rate around a hundred Mbps and some systems going up to 1 Gbps [51]. However, such systems do not reach more than a few dozen metres in terms of range in the best conditions, which means clear water without particles. As soon as turbidity comes in the way, this kind of system gets quickly inefficient. Moreover the range achievable depends on both the colour of the light emitted and the colour of the medium. Hence this technology is not suitable for our application where the robot might be travelling in clear blue water between the plants and manoeuvre in a greenish water when coming around the structures where algae develop easily.

With no efficient solutions to control a robot wirelessly, the Autonomous Underwater Vehicle (AUV) have been designed. This particular type of vehicle has been investigated to perform missions where no real time communication can be set up. They can achieve achieve particular tasks without any kind of supervision. They have been involved in wide range of applications such as hydrography, oceanography, exploration [75] but also military related operations or even archaeology [73]. With the development of offshore wind farms and sensor fields, one of their basic mission is data muling as envisioned in the USMART project [92]. It consists in gathering data from a certain point let us say a sensor attached to floating wind turbine that measures salinity, pressure or any kind of value and bring it

back to some mother station where the data will be analysed.

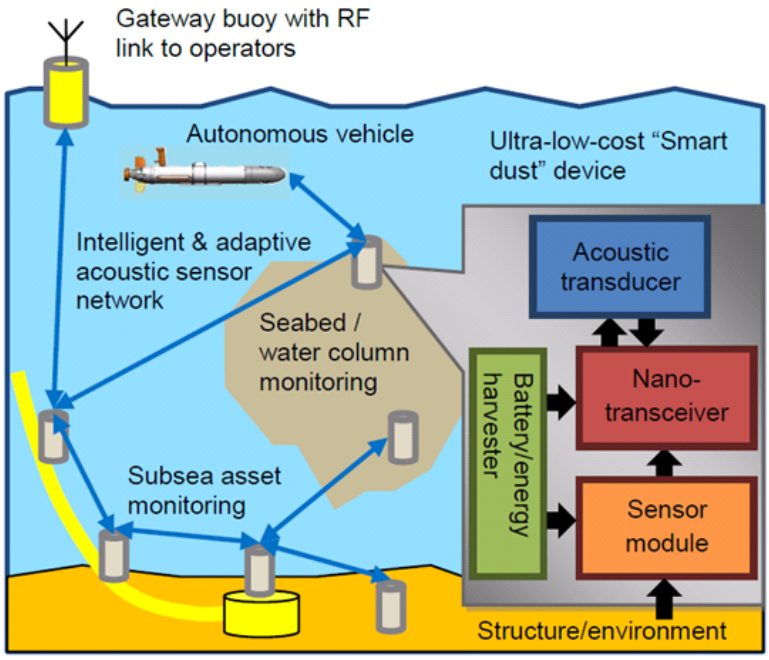


Figure 1.4: Data muling scheme of operations (USMART https://research.ncl.ac.uk/usmart).

As they need to evolve unsupervised, these robots are equipped with numerous sensors themselves in order to sense their environment and then act accordingly to achieve their task. These can be cameras [77], sonars, pressure sensor, magnetic sensors or even electric sensors [15] To perform successfully, one of the critical point to address is the localisation of such vehicles. Indeed if the mission is to gather data from a particular sensor, the robot must be able to reach the defined sensor.

This thesis has been funded by Kopadia, a French Start-up company based in Nantes France, which is specialised in AUV operations. The robot we will consider and work with, is one of theirs: the Folaga presented in Figure 1.5. The Folaga is a low cost robot that is only equipped with an inertial measurement unit and a magnetic compass. This robot being highly modular, it is easy to add new sensors under the form of mountable payload. However, each new payload will have to be powered and adds to the system weight and length. Therefore we may avoid to use multiple payloads at the same time.

1.3 The localisation problem

As mentioned in the previous section, the challenging problem of localisation in the field of AUVs is crucial. Indeed, as we have seen just before, radio signals do not travel though water. As they are usually used to carry information in the robotics world, a whole class of sensors and communication devices are not available to tackle our problem. Especially, any variant of Global Navigation Satellite System (GNSS) system, among them the American

Figure 1.5: The Folaga

Global Positioning System (GPS) is not available as reliable source of positioning. This is similar to the situation of a robot navigating indoors with the added constraint that underwater robots are also deprived from the classic radio communications. These can be of great help for the vehicle to gather information on its current localisation. Therefore other approaches have been investigated to face the lack of information induced by this class of devices unavailability.

1.3.1 Dead reckoning

The first method that does not rely on external measurements is to measure the forces applied on the system. By a simple mean of integration it is possible to estimate the displacement of the vehicle between two measurements. This particular method is called dead reckoning. Numerous sensors are available to perform these measurements. Accelerometers are designed to measure the linear acceleration while gyroscopes return the angular acceleration. In addition to these two, the gyrocompass, which is a magnetic sensor returns the north direction. Fusing the information coming from these three types of sensors, it is possible to compute an estimation of the 3D orientation of the vehicle. Data fusion is often performed by one of the non linear flavour of the Kalman Filter, such as the Extended Kalman Filter (EKF) or the Unscented Extended Kalman Filter (UEKF). This machinery forms the Attitude and Heading Reference System (AHRS). Usually, unmanned vehicles embark an upgraded version of the AHRS which is called an Inertial Navigation System (INS). As in the AHRS, the INS is equipped with sensors for all 6 degrees of freedom (*i.e.* 3 accelerometers and 3 gyroscopes) and a gyrocompass. But in addition to returning an estimation of the vehicle orientation, it also gives an estimation of the position of the robot and its velocities. This kind of system comes in wide range of prices and therefore quality. The main drawback of such systems is that biases are introduced during the integration step. Depending on the quality of the INS, these biases may be quite significant; resulting in speed and position values with a large uncertainty and/or false results. This phenomenon is called drift.

To counter this drifting effect, several options have been investigated. The first one is to ascend back to the surface periodically in order to recalibrate the INS using the GNSS

system. Depending on the quality of the INS and the margin of precision one would like to keep, the length of the period underwater can vary greatly. Creating bathymetric maps demands a precision of 2m in its highest rank (special order S-44). Even a military grade INS would not meet the requirements for more than an hour. However the price of such grade A equipment is not affordable for multitude of robots. Therefore it is not suitable for offshore plants monitoring. Moreover these systems are huge and does not fit with small AUVs.

A second option is to add a Doppler Velocity Log (DVL). This equipment measures the speed over ground of the vehicle based on the Doppler effect. It sends four acoustic beams toward the ground to calculate the Doppler effect appearing when it receives the bounces back. Then data fusion is used with the data given by the INS, using again a flavour of Kalman filter, to get a better estimation of the displacement and thus current position of the robot. When an INS and a DVL are coupled together, the system becomes an Aided Inertial System (AINS)

Lastly, the position computed by an INS is always based on a known initial position. Therefore every computed estimation is made relatively to it. Hence if no absolute coordinates are known for this initial position we do not get an absolute localisation of the robot. In the context of monitoring, if the robot detects a leak on a pipe for instance, without absolute reference, it will not be able to place it on a geographical map to solve the issue quickly. Moreover one major problem is, because of the drift effect mentioned earlier, error on this estimation never stops growing with time. Hence navigating using this system alone cannot be considered for long missions. Some examples of AUVs using this navigation system can be found in [16, 105, 122]. But it is always complemented with another method to correct the position estimation error due to the drifting effect and sometimes provide an absolute reference for positioning.

1.3.2 Acoustic positioning

In the previous section, we investigated navigation using only sensors measuring proprioceptive characteristics. Another method to localise an autonomous vehicle is to use an external system. In the scope of AUVs such systems are acoustic beacons. In this area, two different configurations have been thoroughly investigated.

1.3.2.1 Ultra Short Baseline

The first one is the Ultra Short Baseline (USBL) positioning system based on four hydrophones to listen to acoustic signals and a transmitter that has to be mounted on a surface vehicle (see Figure 1.6 for a schematic). The transmitter sends pings to the transponder mounted on the underwater vehicle to emit. This emission will be received by the four hydrophones, allowing the USBL system to locate the vehicle in its coordinate frame. The surface vehicle having access to a GNSS system and an INS, it is then possible to calculate the absolute position of the AUV we want to locate. This system is quite convenient to use and relatively cheap but has two major drawbacks:

- It needs a surface vehicle with the ability to have a precise position measurement.
- The underwater vehicle never knows its position with the USBL system alone, as everything is calculated on the surface vehicle after receiving the response from the transponder. Thus another communication system must be used to send the position back to the AUV if required.

Moreover, the range of such a system can be low depending on the mission that has to be performed. On average, a maximum distance of 5km can be obtained. In the case of distributed underwater robotics where the aim is to cover a maximum space for exploration, this range limitation can become an issue.

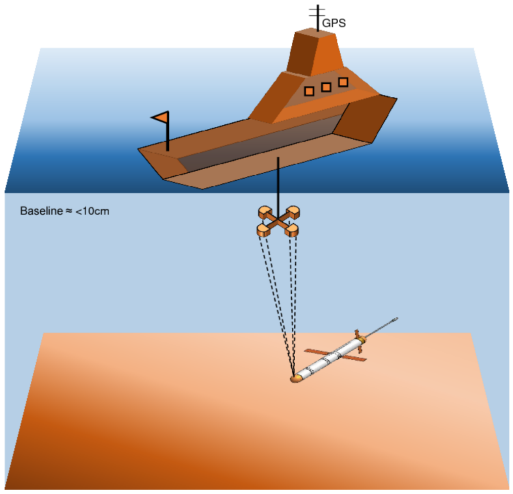


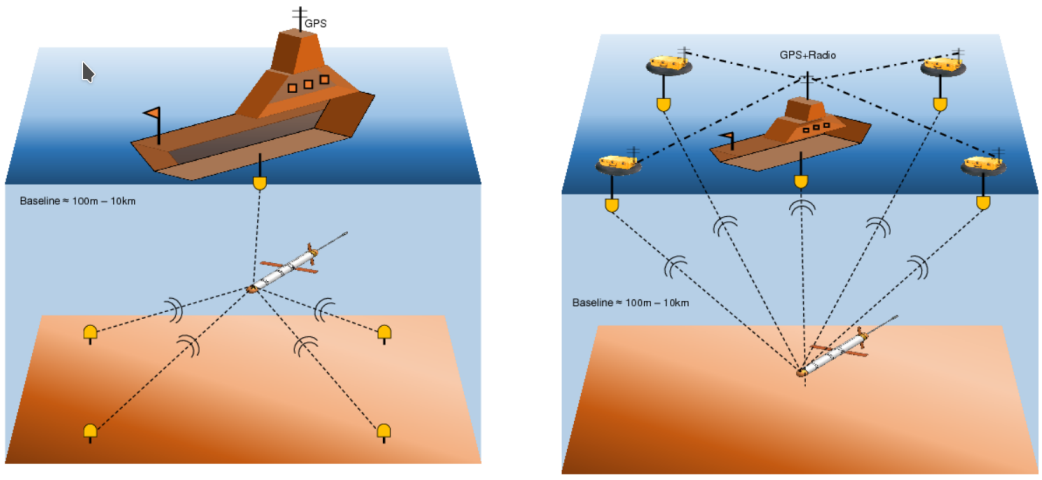
Figure 1.6: USBL configuration [43].

1.3.2.2 Long Baseline

The second configuration is the Long Baseline (LBL) system. Unlike the USBL system, the LBL system uses several transponders with a precisely known position, either fixed on the seabed or in more recent systems, attached to buoys with or without an anchor as shown in Figure 1.7. Each fixed transponder waits for an emission coming from the underwater vehicle. When it receives one it transmits a response with its position. Using the time elapsed between the moment it sends a call and the time it receives an answer, the AUV is able to measure the distance between itself and the beacon. Using measurements from several beacons, the AUV is then able to calculate its position and can use it for navigation. This is called triangulation or trilateration.

At least three responses are needed to have a precise position in 2D; at least four responses for a 3D position. That is why there are usually more than four beacons deployed to ensure that the AUV will not be blind because of the geometry of the environment. Moreover, redundant information gives better results in case the number of responses received is higher than the one needed. This method allows a better range and is generally more robust, but

it is far more costly as the beacons have to be fixed on the seabed and possibly gathered at the end of the mission. Several variants have been investigated to reduce the number of beacons needed such as Sparse-LBL and Virtual-LBL [76].



(a) Classical LBL system (b) Mobile LBL system

Figure 1.7: LBL systems configurations [43]

In the context of constant monitoring, the USBL and its need for a surface vessel is not ideal. On the other hand, placing beacons on already positioned plant seems feasible. Therefore, the LBL system will be preferred. Moreover, Kopadia developed a system of floating beacons which embark a GNSS antenna on its air side and an acoustic beacon on the end that stays underwater. Hence, it can get its absolute position while communicating with an AUV navigating underwater. Using multiple of these it can be used as a mobile LBL system.

1.4 Research approach

1.4.1 Solving the localisation problem

The main objective of this work will be to solve the localisation problem in the case of a low cost AUV as the Folaga presented earlier. Its INS being a low price one, it is highly sensitive to the drift mentioned in Section 1.3.1. Therefore to help it estimating the robot position, we will use its dynamical model with numerical integration. Moreover such systems need to have a precise initial condition to work properly. However this is very unlikely in an on-field operation. Therefore there will already be an error at the initialisation of the estimation algorithm. Hence this work will aim at producing a method that is robust to the uncertainty we can have on the initial condition of the system. Moreover we will consider that we may have during the course of operations, some range measurements with the floating beacons that the robot vehicle may come close to. This sums up all the information we may have with our robot.

1.4.2 Constraint Satisfaction problem

Each new information on the system in the localisation problem can be written under the form of an equation. Its dynamics can be rewritten under the form of a state equation (see Section 2.2.1), the different range measurements are simple distance calculations. All these pieces are information on where the vehicle may be but also constraints on where it cannot be. This set of equations forms what is called a Constraint Satisfaction Problem (CSP) (see Section 2.3.4). Therefore solving the localisation problem comes down to solving a CSP.

This particular class of problem has been studied in various context, using several type of algorithms [70]. In this thesis, we will use a set-membership approach which has been proven suitable to solve this kind of problems.

1.4.3 Set-Membership methods

Most of the methods we can find in the literature when it comes to position estimation will be based on probabilities. These probabilistic methods will return a result and a covariance matrix which shows how reliable the result is and what is the error to be expected. Among them we can cite the Kalman Filter (KF) and its different variants, the Particle Filter (PF) and so on .... These methods are only probabilistic and the results are not guaranteed. This can be a major issue when notions of safety arise. When dealing with complex systems such as robots and critical systems like energy plants, safety is essential. Therefore the need for a guarantee in our computations. To face it, we have chosen to use set-membership methods. This kind of approach has shown strong abilities to deal with non linear problems and uncertainties. Unlike probabilistic methods, the result obtained is a *solution set* the actual solution is part of. Therefore all computations made will not remove feasible solutions, which guarantee our result. The shortcoming of this, is that it may induce pessimism. This means that the solution set may contain a large number of feasible solutions in addition to the actual one. One other major difference of set-membership methods with the probabilistic approach, is that all computations are deterministic. Thus, if the input is unchanged the output stays the same, if the computation steps are identical, which is not necessarily the case with probabilistic methods.

1.4.4 Thesis outline

This thesis comes within the scope of Kopadia's development. Hence, it focuses on improving the localisation of a low-cost AUV equipped with cheap sensors. Therefore their performances are relatively low compared to the actual standards. This thesis is at the crossroads of three different fields of work which are interval analysis, guaranteed integration and Lie groups. The three chapters following this introduction cover their presentation.

After introducing some tools to study dynamical systems, from which robots are part of, Chapter 2 focuses on interval analysis. After presenting the basics of this set-membership approach, the different methods provided by interval analysis to handle dynamical systems problems are exhibited.

Then Chapter 3 presents the notion of guaranteed integration. The different conventional methods to perform it are shown. In this chapter, is also presented a small contribution of this thesis which is the integration of one of the most efficient guaranteed integration tool into the library used to perform the different computations throughout this work.

To end this first part, Chapter 4 introduces the field of Lie groups. We will intentionally stay on the edges of this field and only present the tools needed later in this work. Though being extremely powerful, Lie groups are very abstract, thus we will present them in the context of differential equations. In this chapter is presented one of the main contributions of this thesis named the *transport function*. As the reader may not be acquainted with one or more of these fields, the different notions covered are illustrated with many examples and some pieces of codes are given and explained for the readers to try the different tools used in this thesis by themselves.

Chapter 5 brings the main contribution of this thesis. Using the different elements introduced in the previous chapters, a new integration method has been developed. It is designed to be robust to the uncertainties mentioned earlier. It is illustrated through four examples from the simplest theoretic one to a problem closer to robotics.

Chapter 6, then shows different areas where the method developed in Chapter 5 could be applied with examples. Finally, Chapter 7 concludes this work by summarising the different contributions of this thesis and presenting the trails one may follow to pursue it.

CHAPTER 2

MODELLING ROBOTS AND SET-MEMBERSHIP METHODS

Contents

2.1 Introduction

We recall that the objective of this work is the planning of a mission of a robot and its localisation after the operation. To reach such goals, we will need a *model* of our system in order to study its evolution through time given an initial state. We tackle the subject in section Section 2.2. However, as accurate as our model can be, we still have to deal with uncertainties coming from different sources. Among them, external perturbations that could occur while our system is in operation. An example, in our case, could be a water current applied to our robot during a mission. Another source of uncertainty is the accuracy of our sensors. As good as they can be, none are perfect and will introduce uncertainties in our model. Lastly, the computational power available and the representation of numbers in computers will hinder the efficiency of our model.

The main contribution of this thesis presented in Chapter 5 is at the crossroads of three different mathematical fields, interval analysis and tubes presented in Section 2.3 and Section 2.4 respectively, guaranteed integration explained in Chapter 3 and Lie theory developed in Chapter 4. To keep this presentation concise, and help the reader understanding, we will focus on the different tools needed from each field. Hence a lot of pointers to references will be made to more exhaustive explanations throughout these chapters.

2.2 Modelling a mobile robot

A mobile robot is an unmanned mechanical system able to move in its environment may it be ground, air, water or even space. It can be autonomous or remotely operated but must be able to sense its surroundings using sensors, act on it with its actuators and embark some kind of intelligence to link both. As it is a mechanical system, it is subject to the laws of physics and thus can be described with mathematical equations

2.2.1 Dynamical System

Mathematically speaking, a mobile robot can be seen as a dynamical system following a set of equations describing its evolution in time. A general definition of a dynamical system can be found in [42, Chapter 1]

Definition 2.1 (Dynamical system)**.** A dynamical system is a function $\phi : T \times \mathcal{S} \to \mathcal{S}$ which follows the properties below:

1. T is either $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ or $\mathbb{R}^+$. $t \in T$ will be the evolution parameter and T the time set;
2. $\mathbb{S}$ is a non empty set. It is the state space set;
3. $\phi(0, .)$ is the identity function, *i.e.* $\forall \mathbf{x} \in \mathcal{S}, \phi(0, \mathbf{x}) = \mathbf{x}$;
4. For any $\mathbf{x} \in \mathbb{S}$, and $t, \tau \in T, \phi(t, \phi(\tau, \mathbf{x})) = \phi(t + \tau, \mathbf{x})$.

Remark 2.1. From point 3 of the definition above, we can notice that it is impossible for the system to evolve in its state space instantaneously.

Remark 2.2. Coming from item 4, the future state of the system always depends on its current state and it is assumed that the system dynamics stays the same during its evolution.

There exist three different types of dynamical systems, continuous, discrete and hybrid. In this thesis we will only deal with continuous time systems. Hence only this type will be developed further. The interested reader can refer to [106] for discrete systems and [40, 41, 109] for information on hybrid ones.

Let us introduce some notations and vocabulary that we will use throughout this work:

1. When T is equal to $\mathbb{R}$ or $\mathbb{R}^+$ the system is called *continuous-time*, on the contrary to discrete-time systems where $T = \mathbb{N}$ or $\mathbb{Z}$.
2. The state vector of the system will be called $\mathbf{x}$.
3. We will denote the initial condition by $\mathbf{x}_0$ the state of the system at time $t = 0$.

2.2.2 Continuous-time system

The subclass of dynamical systems we will deal with gathers continuous-time systems. Also called *smooth dynamical systems*, a definition of them can be found in the books of Hirsch [53, Chapter 7] and Grass [49, Chapter 2].

2.2.2.1 Definition

Definition 2.2 (Continuous-time system)**.** A smooth dynamical system on $\mathbb{R}^n$ is a continuously differentiable function $\phi : \mathbb{R} \times \mathbb{R}^n$, where $\phi(t, \mathbf{x})$ satisfies the following properties:

1. $\phi(0, .)$ is the identity function, *i.e.* $\forall \mathbf{x} \in \mathbb{R}^n, \phi(0, \mathbf{x}) = \mathbf{x}$
2. For any $\mathbf{x} \in \mathbb{R}^n$, and $t, \tau \in \mathbb{R}, \phi(t, \phi(\tau, \mathbf{x})) = \phi(t + \tau, \mathbf{x})$

The careful reader will have noticed that if we have an analytical expression for ϕ and an initial condition $\mathbf{x}_0$, it is trivial to compute the state $\mathbf{x}(t)$ at time $t \in T$. Unfortunately, in most cases, there is no such expression available.

Remark 2.3. In the rest of this thesis, we will often simplify $\phi(t, x)$ by $\phi_{\mathbf{x}}(t)$ if $\mathbf{x}$ is a fixed parameter or by $\phi_t(\mathbf{x})$ if t is a fixed parameter.

2.2.2.2 Relationship between ordinary differential equations and continuous time systems

Since the introduction of differential and integral calculus by Newton and Leibniz, dynamical systems have been studied in many different fields from physics [115] to chemistry, from engineering to biology [57] and even economics [125] or sociology [111] to describe various types of phenomena. If the systems described are very different from one another, they all have in common the use of differential equations (partial or ordinary) to represent such systems. In the case of a robot, one would use classical mechanical work which would lead to Equation (2.1)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)), \tag{2.1}$$

where $\mathbf{f} \in \mathcal{C}^k(\mathbb{R}^n)$, $t \in \mathbb{R}$ represents the time and the vector $\mathbf{x} \in \mathbb{R}^n$ is the state of the system. This equation describes the systems behaviour and only depends on the current state of the system. This type of differential equation is called *autonomous* which means that the independent variable t does not occur explicitly in the expression of $\mathbf{f}$. Thus the expression is often simplified into:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}). \tag{2.2}$$

In numerous robotic examples, $\mathbf{f}$ also depends on a variable $\mathbf{u}(t) \in \mathbb{R}^m$ which is an external input that often represents, in a robotics context, an actuator command.

However, if Equation (2.1) alone describes the general behaviour of a system, it fails to give a particular solution. This is the reason why it is usually associated with an initial condition to form the system 2.3

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)), \quad t \in [t_0, t_f] & \text{(2.3a)} \\ \mathbf{x}(t_0) = \mathbf{x}_0 \in \mathbb{R}^n & \text{(2.3b)} \end{cases}$$

The system (2.3) is called an Initial Value Problem (IVP). A definition can be found in [49, Chapter 2].

Definition 2.3 (Ordinary Differential Equation (ODE), IVP, Solution)**.** The Equation (2.3a) is called an ODE and the System (2.3) is called an IVP for the initial condition given by Equation (2.3b).
The function $\mathbf{x} : [t_0, t_f] \to \mathbb{R}$ is called a *solution* to the IVP (2.3) if $\mathbf{x}(t)$ satisfies (2.3a) for all $t \in [t_0, t_f]$ and $\mathbf{x}(t_0) = \mathbf{x}_0$.

The use of IVPs is widely spread among the field of dynamical systems, especially for autonomous systems. In such a configuration, the Picard-Lindelöf theorem proves the existence and the unicity of the solution $\phi(t, x_0)$ [53, Chapter 7].

Theorem 2.1 (Picard–Lindelöf)**.** *Consider the IVP*

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \in \mathbb{R}^n \end{cases}. \tag{2.4}$$

Suppose that $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$, then there exists a solution for this IVP and this solution is unique

The interested reader can find a proof of this theorem in [53, Chapter 17]. The solution $\phi(t, x_0)$ obtained predicts the future state of the system initialised at $\mathbf{x}(t_0) = \mathbf{x}_0 \in \mathbb{R}^n$ for a time $t \in [t_0, t_f]$.

Remark 2.4. As mentioned earlier, the analytical expression of the flow ϕ is not always available. In this case, there exist numerous methods to compute numerical solutions, (the simplest one being Euler method which is rarely used nowadays as it is very weak) and even guaranteed ones as we will see in Chapter 3. These methods are used to evaluate ϕ_t at a specific time t for a defined initial condition $\mathbf{x}_0$.

Remark 2.5. In some particular situation, the calculated solution may only be valid locally, in that case we deal with *local dynamical systems*.

Example 2.1 (Continuous time system)**.** We will end this section with a short example of continuous dynamical system. Let us take the system defined in [54] as follows:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} -x_1^3 - x_1 x_2^2 + x_1 - x_2 \\ -x_2^3 - x_1^2 x_2 + x_1 + x_2 \end{pmatrix}. \tag{2.5}$$

It is possible to represent Equation (2.5) as a *vector field* in the state space, here $\mathbb{R}^2$, with a time set being $\mathbb{R}$. Throughout this work, for ease of understanding, we will not differentiate the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ and its associated vector field $\mathbf{f}$. In Figure 2.1, the vector field associated with Equation (2.5) is drawn. The flow ϕ associated with $\mathbf{f}$ also has a geometrical representation. It can be interpreted as a trajectory going through a specific point, the initial condition for instance. Several trajectories have be added with different conditions at $t = 0$ on Figure 2.1. The careful reader may notice different notations. $\phi(t, \mathbf{x}_i)$ denotes the trajectory (the curve), passing through the initial condition $\mathbf{x}_i$, when $\phi(3, \mathbf{x}_i)$, denotes the point of the trajectory at time $t = 3$ for a given initial condition $\mathbf{x}_i$.

2.3 Interval Analysis

The previous section gave us tools to model our robot thanks to dynamical systems study. We will use them later in this work. We now need a way to handle the uncertainties that may appear when dealing with it. Indeed, as correct and close to the true system our model is, there will still be uncertainties coming from different sources, the model itself, external perturbations, computing errors to name a few. In this thesis, interval analysis will be used to tackle this issue.

2.3.1 Background

2.3.1.1 Origins

The mathematical field of interval analysis has been originally developed to address the problem coming from rounding errors when calculating with computers [33, 113]. In a machine, there is only a finite set of numbers available to the user. We will call this set $\mathcal{S}_{comp}$ where $\mathcal{S}_{comp} \subset \mathbb{R}$. The reason for this, is the representation of numbers in computers

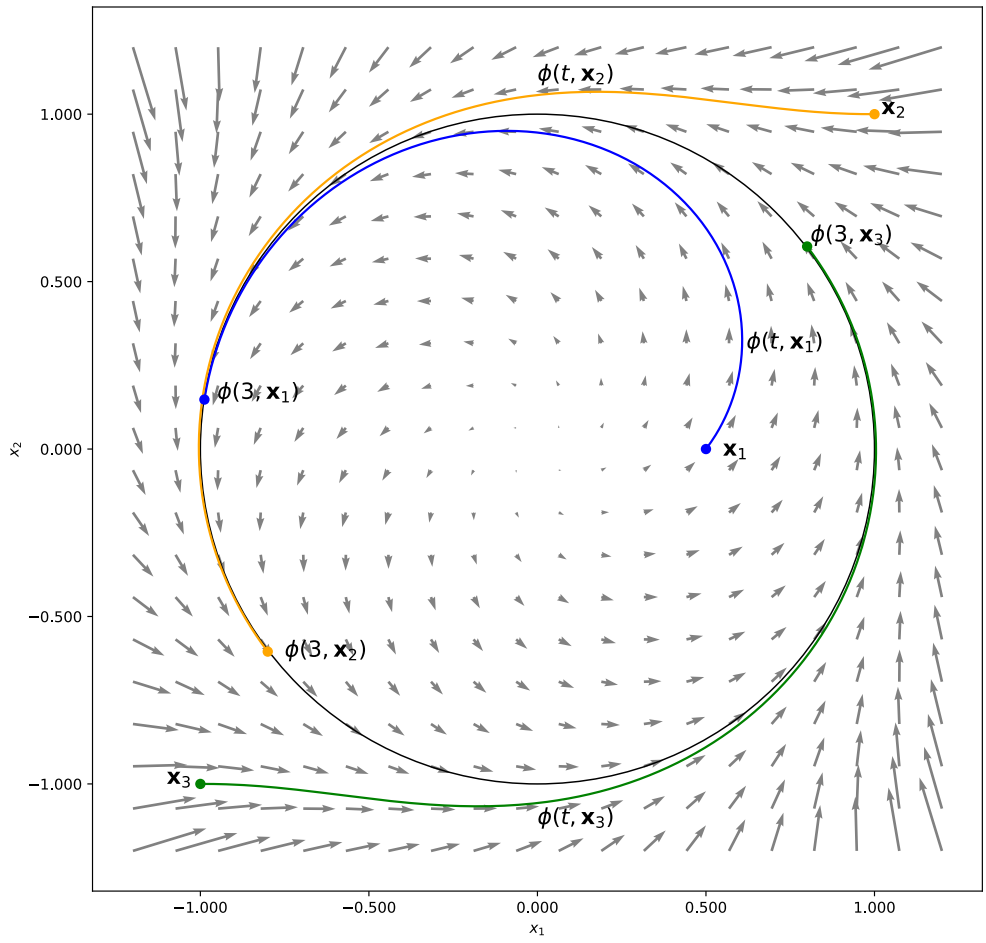


Figure 2.1: Vector field associated with Equation (2.5). A few trajectories passing though different initial conditions are represented

memory under the form of bytes which have a finite size (64 bits for most computers nowadays). A simple way to verify this, is to open a simple Python interpreter in the console and try perform a simple calculation for instance 2.1+2.7. One would expect the result to be 4.8. Unfortunately for us, it is not as you can see in Figure 2.2.

```
Python 3.8.5 (default, Jul 20 2020, 23:11:29)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(2.1+2.7)
4.800000000000001
```

Figure 2.2: Round-off error

Hence the issue when trying to compute models using numbers in $\mathbb{R}$. This rounding error can lead to huge numerical error when it is carried from one computation to another (see [91]). To address this problem, interval analysis represents a real number x as a combination of two bounds a and b such that

$$x \in [a, b], \ a, b \in \mathbb{R}.$$

We often denote the interval $[a, b]$ as $[x]$ since it encloses the value of x. a and b are called lower bound and upper bound respectively. It is now possible to represent x as the closest interval containing it using

$$x \in [a, b], \ a, b \in \mathcal{S}_{comp},$$

where a and b are the values immediately below and immediately above x respectively when $x \notin \mathcal{S}_{comp}$. If $x \in \mathcal{S}_{comp}$ then $a = b = x$. In [90], Moore extended the concept of interval analysis to bound the effect of errors from all sources, including approximation errors and errors in data.

Remark 2.6. This approach is called *set-membership* approach as values are enclosed in a set to deal with them. It has been widely used to solve nonlinear problems [31, 58, 74, 87]. This is opposed to the widely used *probabilistic* approach, which represents uncertainties as a probabilistic functions. This other method may be less computationally expensive and its results easier to understand. However it lacks the guarantee in the results we will later need in this work.

2.3.1.2 Applying interval analysis in a robotics context

Interval analysis comes in handy when addressing robotic problems. When dealing with this kind of systems, we often want to prove, for safety reasons, that it is located at one place, that it will reach some state or avoid one, whatever the perturbations. Hence the need for guaranteed computation that interval analysis can handle [116]. Indeed, if we represent the different components of the state vector and the parameters related to the problem as intervals then it is possible to perform guaranteed computation. Therefore one can use numerical results as proof. Interval analysis has been used for various problems in robotics for instance research of stability sets [14], localisation [68],reachability analysis [86] or finding basins of attraction [29].

2.3.2 Interval arithmetic

In the following section we will introduce the different concepts and notations used later in this work related to interval analysis. Most of them are derived from [63, Chapter 2]

Definition 2.4 (Interval)**.** An interval denoted $[x]$ is a closed and connected subset of $\mathbb{R}$. We denote by $\mathbb{IR}$ the set of all intervals.

Example 2.2. Here are a few examples of intervals:

1. $[1, 2]$ is an interval,
2. $[\frac{1}{3}, 4]$ is an interval,
3. $[\sqrt{2}, \pi]$ is an interval,
4. $\{1\}$ is an interval. It is called a *degenerate* interval,
5. $\emptyset$ is an interval,
6. $[-\infty, \infty]$ is an interval.

As mentioned earlier, an interval possesses a lower bound $\text{lb}([x])$ and an upper $\text{ub}([x])$ bound. The former is often denoted $\underline{x}$ or x^- and the latter $\overline{x}$ or x^+ in the literature. We will use the second notation throughout this work. Those are defined as follows:

$$x^- = \text{lb}([x]) \triangleq \sup \{a \in \mathbb{R} \cup \{-\infty, \infty\} \,|\forall x \in [x], a \leq x\} \tag{2.6a}$$
$$x^+ = \text{ub}([x]) \triangleq \inf \{b \in \mathbb{R} \cup \{-\infty, \infty\} \,|\forall x \in [x], x \leq b\} \tag{2.6b}$$

Remark 2.7. The symbol "$\triangleq$" means "is defined to be" throughout this work

2.3.2.1 Sets operations

As intervals are sets, the different set operations can be applied such as intersections and unions. The intersection operation can be directly extended to intervals and is defined by:

$$[x] \cap [y] \triangleq \{z \in \mathbb{R} | z \in [x] \text{ and } z \in [y]\} \tag{2.7}$$

One should be careful when applying the union operation. When using sets the union is defined as follows:

$$[x] \cup [y] \triangleq \{z \in \mathbb{R} | z \in [x] \text{ or } z \in [y]\} \tag{2.8}$$

To be consistent with Definition 2.4, the result of the operation should be connected. That is why the *interval hull* of a subset $\mathbb{X}$, denoted $[\mathbb{X}]$ has been defined. It is the smallest interval containing $\mathbb{X}$. For instance the interval hull of $[-4, -2] \cup [1, 6]$ (written

$[[-4,-2] \cup [1,6]])$ is equal to $[-4,6]$. Whence the *interval union* denoted by $[x] \sqcup [y]$ has been defined as

$$[x] \sqcup [y] \triangleq [[x] \cup [y]] \tag{2.9}$$

In the same way *interval deprivation* has also been defined as $[\backslash]$:

$$[x][\backslash][y] = [[x] \setminus [y]] = [\{x \in [x] | x \notin [y]\}] \tag{2.10}$$

Example 2.3 (Sets operations)**.** Here are a few examples of extended sets operations applied to intervals

- $[2,6] \cap [4,9] = [4,6]$ is an interval
- $[-4,-2] \cup [1,6]$ is not an interval
- $[-4,-2] \sqcup [1,6] = [-4,6]$ is an interval
- $[4,7][\backslash][5,9] = [4,5]$
- $[2,10][\backslash][5,8] = [2,10]$

2.3.2.2 Extension of classical operators

As the primary objective of intervals is to represent values of $\mathbb{R}$, all arithmetic operations have also been extended to intervals. For each classical binary operator $\diamond \in \{+,-,\times,/\}$:

$$[x] \diamond [y] = [\{x \diamond y | x \in [x], y \in [y]\}]. \tag{2.11}$$

For addition and subtraction, it only consists in applying the operator on the bounds. When dealing with multiplication and division, one should be careful especially if $\{0\}$ is included in one of the terms. For further information on implementation, the reader may refer to [63].

Example 2.4 (Arithmetic of $\mathbb{R}$)**.** A few examples of classical operations extended to intervals are given below

1. $[1,3] + [2,6] = [3,9]$
2. $[1,3] - [2,6] = [-5,1]$
3. $[1,3] * [2,6] = [2,18]$
4. $[1,3] * [-2,6] = [-6,18]$
5. $[1,3]/[2,6] = [\frac{1}{6}, \frac{3}{2}]$
6. $[1,3]/[-2,6] = [-\infty,\infty]$

Complex operators such as trigonometry operators, exponential, logarithm and so on have also been extended to intervals. Therefore they are not limited to linear operations and can be used when addressing non linear problems.

Remark 2.8. It is possible to stack intervals to form interval vectors often called *boxes* denoted $[\mathbf{x}]$. The latter denomination will be used throughout this work. These boxes are called *axis-aligned* as each interval of the box is a subset of one dimension of $\mathbb{R}^n$ (see Figure 2.3) All operations presented in Section 2.3.2.1 and Section 2.3.2.2 can be applied to boxes. In this case, they are applied dimension-wise.

Remark 2.9. The state of a robot will be often encapsulated in a box $[\mathbf{x}]$, each dimension representing a state component.

Example 2.5 (boxes)**.** We introduce two boxes $[\mathbf{x}] = [0,3] \times [2,4]$ and $[\mathbf{y}] = [2,5] \times [1,3]$

1. $[\mathbf{x}] + [\mathbf{y}] = \begin{pmatrix} [2,8] \\ [3,7] \end{pmatrix}$
2. $[\mathbf{x}] \cap [\mathbf{y}] = \begin{pmatrix} [2,3] \\ [2,3] \end{pmatrix}$ (see Figure 2.3)

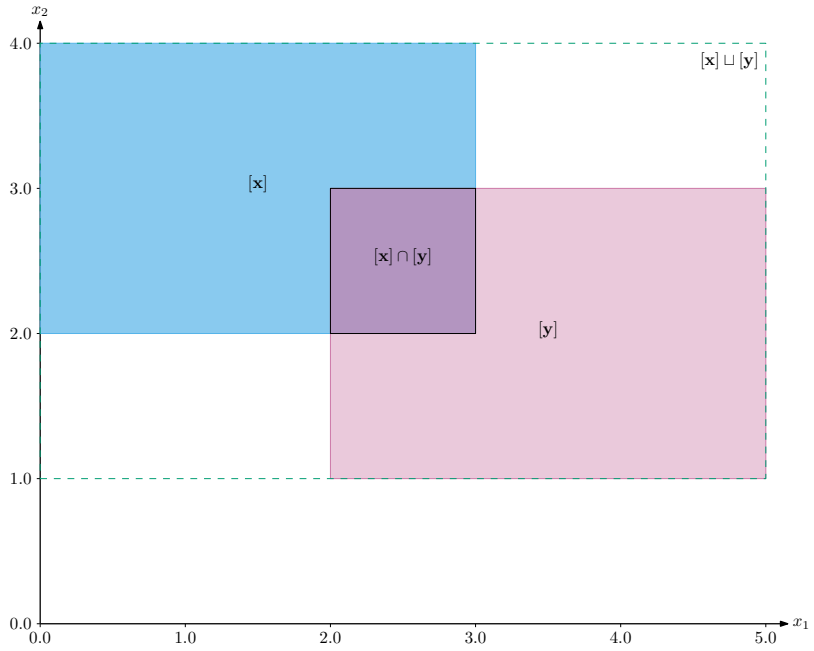


Figure 2.3: Sets operations applied on boxes.

2.3.3 Inclusion functions

2.3.3.1 Extension of functions to intervals

In the previous section, we have seen that classical operators of $\mathbb{R}$ ($\mathbb{R}^n$ in case of boxes) could be extended to intervals. In the same way, interval functions are the extended form

of real functions to this field.
Consider a box $[\mathbf{x}] \in \mathbb{IR}^n$ and a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. The image of $[\mathbf{x}]$ by $\mathbf{f}$ is given by:

$$\mathbf{f}([\mathbf{x}]) = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\}. \tag{2.12}$$

The image set obtained may be an axis-aligned box especially with elementary functions but generally it is not the case (see Figure 2.4). Hence an interval counterpart has been defined, called *inclusion function*.

Definition 2.5. $[\mathbf{f}] : \mathbb{IR}^n \to \mathbb{IR}^m$ is an inclusion function for $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ if

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) \subseteq [\mathbf{f}]([\mathbf{x}]). \tag{2.13}$$

An inclusion function returns a box enclosure of the image of $[\mathbf{x}]$ by $\mathbf{f}$. It is usually easier to compute as it can be made combining the interval operators presented in the previous sections. Moreover the image is a box, thus the results can be used again for interval computations. However, because the inclusion function returns a box, some elements of $[\mathbf{f}]([\mathbf{x}])$ may not have a preimage by $\mathbf{f}$. This phenomenon is called the *wrapping effect* [34, 88]. An inclusion function can have several properties presented below:

- An inclusion function is said *thin* if the image of any degenerate box $[x] = x$ is also degenerate *i.e* $[\mathbf{f}](\mathbf{x}) = \{\mathbf{f}(\mathbf{x})\}$
- It can be *inclusion monotonic* if: $[\mathbf{x}] \subset [\mathbf{y}] \Rightarrow [\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{f}]([\mathbf{y}])$
- It is *minimal* if $\forall [\mathbf{x}], [\mathbf{f}]([\mathbf{x}])$ is the smallest box enclosing $\mathbf{f}([\mathbf{x}])$. In that case it is denoted $[\mathbf{f}]^*$. For a given function $\mathbf{f}$, there exists an infinity of inclusion functions, but only one can be minimal. By definition the minimal inclusion function is the one inducing the least wrapping effect.
- $[\mathbf{f}]$ is *natural* if each variable x_i of $\mathbf{f}$ is substituted by an interval variable such that $x_i \in [x_i]$ and each real operator by its interval counterpart to create $[\mathbf{f}]$. The natural inclusion function is the simplest one to find but may not be minimal.

Example 2.6. (Natural inclusion function) Consider the function $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x^3 - x^4$. The natural inclusion function associated with it is

$$\begin{array}{rccc} [f] : & \mathbb{IR} & \to & \mathbb{IR} \\ & [x] & \mapsto & [x]^3 - [x]^4 \end{array} \tag{2.14}$$

Example 2.7. (wrapping effect) As shown in Figure 2.4, there is an infinity of inclusion functions for one given function and some will yield better enclosure than other. This induced pessimism has been studied, and it has been noted that different analytical expressions often lead to far different performances in terms of sharpness of the enclosure when turned into inclusion functions. The simplest example that comes to mind is the function $f(x) = x - x = 0$. If we consider its associated natural inclusion function we obtain:

$$\begin{aligned} [x] - [x] &= [\{x_1 - x_2 | x_1 \in [x], x_2 \in [x]\}] \\ &= [x^- - x^+, x^+ - x^-] \end{aligned}$$

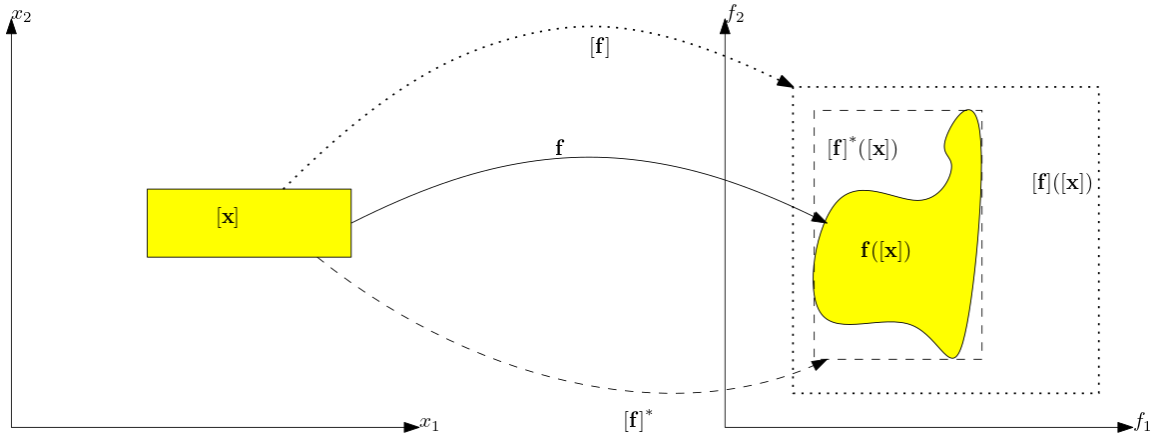


Figure 2.4: Here is represented a box $[\mathbf{x}]$ and its image set by function $\mathbf{f}$. In addition, the image by the natural inclusion function of the latter is represented as a doted box. Then the minimal enclosure obtained thanks to the minimal inclusion function is drawn as a dashed box.

If the interval is not degenerate, it is easy to figure out that the result is far from the one expected. For instance if we set $[\mathbf{x}] = [-1, 1]$, then we obtain $[x] - [x] = [-2, 2]$

It has been showed that this problem appears when a variable appears several times in the analytical expression. Different factorisation techniques have been developed to find an analytical expression that limits the wrapping effect [3, 17, 107].

2.3.3.2 Centred form

We will now introduce the notion of centred form of an interval function as they will be of use in Section 3.4.5. This notion has been introduced in [90] and Ratschek provided an explicit formula for them in [100]. A general defintion can be found in [8].

Definition 2.6. Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ and a vector $\mathbf{x} \in \mathbb{R}^n$. Let us introduce $L : \mathbb{IR}^n \times \mathbb{R}^n \to \mathbb{R}^n$, a Lipschitz function for $\mathbf{f}$ over $[\mathbf{x}]$ *i.e*

$$\forall \mathbf{x}, \mathbf{c} \in [\mathbf{x}], \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{c}) \in L([\mathbf{x}], \mathbf{c})(\mathbf{x} - \mathbf{c}) \tag{2.15}$$

Then a centred form $[\mathbf{f}_c]$ of $\mathbf{f}$ on $[\mathbf{x}]$ with a centre $\mathbf{c}$ is given by:

$$[\mathbf{f_c}]([\mathbf{x}]) = \mathbf{f}(\mathbf{c}) + L([\mathbf{x}], \mathbf{c})([\mathbf{x}] - \mathbf{c}) \tag{2.16}$$

Proposition 2.1. *A centred form* $[\mathbf{f_c}]$ *of* $\mathbf{f}$ *is an inclusion function for* $\mathbf{f}$, i.e

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) \subset [\mathbf{f_c}]([\mathbf{x}]) \tag{2.17}$$

Proof.

$$\begin{aligned} \forall [\mathbf{x}] \subset \mathbb{IR}^n, \forall \mathbf{x} \in [\mathbf{x}], \mathbf{f}(\mathbf{x}) &= \mathbf{f}(\mathbf{c}) + \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{c}) \\ &\in \mathbf{f}(\mathbf{c}) + L([\mathbf{x}], \mathbf{c})(\mathbf{x} - \mathbf{c}) \end{aligned}$$

Thus,

$$\begin{aligned}\mathbf{f}([\mathbf{x}]) &\subset \mathbf{f}(\mathbf{c}) + L([\mathbf{x}], \mathbf{c})([\mathbf{x}] - \mathbf{c}) \\ &\subset [\mathbf{f_c}]([\mathbf{x}])\end{aligned}$$

□

It should be noticed that the performances, *i.e* the sharpness of the resulting enclosure, of a centred form $[\mathbf{f_c}]$ heavily depends on the choice of $\mathbf{c}$ as shown in [8].

Example 2.8. Consider $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x(1-x)$, $L([x]) = 1 - 2[x]$. Then

$$[f_c]([x]) = c(1-c) + (1-2[x])([x]-c) \ .$$

Let us define c_1, c_2, c_3 such that

$$\begin{aligned}c_1 &= \text{mid}([x]) \\ c_2 &= \text{mid}([x]) - \text{width}([x])/2 = x^- \\ c_3 &= \text{mid}([x]) + \text{width}([x])/2 = x^+\end{aligned}$$

where mid($[x]$) is the midpoint of $[x]$.
Consider $[x] = [0, 4]$, we obtain:

$$\begin{aligned}[f_{c_1}]([x]) &= [-16, 12] \\ [f_{c_2}]([x]) &= [-28, 4] \\ [f_{c_3}]([x]) &= [-16, 16]\end{aligned}$$

Thus the choice of c is of importance when using a centred form. A method to compute an optimal c is given in [8].

Now Definition 2.6 is a general case. In the rest of this thesis when dealing with *centred form* we will use the definition given in [63].

Definition 2.7. Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ such that $\mathbf{f}$ is differentiable over $\mathbb{R}^n$. Then its centred form $[f_c]$ is given by:

$$[\mathbf{f_c}]([\mathbf{x}]) = \mathbf{f}(\mathbf{c}) + [\mathbf{J_f}]([\mathbf{x}]) \cdot (\mathbf{x} - \mathbf{c}) \tag{2.18}$$

where $\mathbf{c} = \text{mid}([\mathbf{x}])$ and $\mathbf{J_f}$ is the Jacobian of $\mathbf{f}$

The main advantage of the centred form is, as the width of $[\mathbf{x}]$ tends toward 0, the enclosure provided by the centred form gets closer to the one given by the minimal inclusion function. Thus it induces very little wrapping effect provided the input box $[\mathbf{x}]$ is small enough. Mathematically speaking, this corresponds to:

$$\lim_{\text{width}([\mathbf{x}])\to 0} \frac{\text{width}([\mathbf{f_c}]([\mathbf{x}]))}{\text{width}([\mathbf{f}]^*([\mathbf{x}]))} = 1 \tag{2.19}$$

2.3.4 Constraint Satisfaction Problems

In robotics many problems can be presented under the form of a CSP. For instance, in a localisation problem, we will look for a position that satisfies all the measurements made with the different sensors our robot is equipped with[59].

A mathematical formalism for CSPs is given in [104, Chapter 2]. A CSP $\langle\sqcup$ is defined by a triple $\langle\sqcup = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is a n-tuple of variables, $\mathcal{D}$ is the set of domains on which these variables are defined and $\mathcal{C}$ is the set of constraints that are applied to these variables. Each element c of the latter set is called a *constraint* and is composed of a pair $\langle scope, rel \rangle$ where *scope* denotes the variables concerned by the constraint c and *rel* the relation linking this variables together, constraining them.

Originally, CSPs where studied over discrete domains [23, 118] and then extended to continuous situations [28, 22, 55]. In this work we will focus on the continuous case.

Proposition 2.2. *It is possible to represent the constraints $c \in \mathcal{C}$ as a multidimensional function $\mathbf{f}$. In that case the formalism comes as follows:*

$$\langle\sqcup = \begin{cases} \mathbf{x} \in [\mathbf{x}] \\ \mathbf{f}(\mathbf{x}) = \mathbf{0} \end{cases}, \tag{2.20}$$

with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ where m is the number of constraints of the problem

Proof. Let us prove that this form of $\langle\sqcup$ defines a CSP:

- $\mathbf{x}$ is the vector of all elements of $\mathcal{X}$
- $[\mathbf{x}]$ represents the domain $\mathcal{D}$
- We introduce $c_i, i \in 1 \dots m$ such that:

$$\forall i, c_i = \langle domain(f_i), f_i(\mathbf{x}) = 0 \rangle .$$

If we define $\mathcal{C} = \{c_1, c_2, \dots c_m\}$, we have $\langle\sqcup = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$. Therefore $\langle\sqcup$ is a CSP.

□

From now on the formalism used in this work to describe CSPs will be the one given in Equation (2.20). We also denote by $\mathbb{S}$ the solution set of $\langle\sqcup$ such that:

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}] | \forall i \in \{1 \dots m\}, f_i(\mathbf{x}) = 0\} . \tag{2.21}$$

Hence, solving $\langle\sqcup$ comes down to finding $\mathbb{S}$. Computing the solution set is recognized as a NP-hard problem in the general case, but for some classes of problems, CSPs can be solved using inference and search methods. One may find more information in [104, Chapter 2]

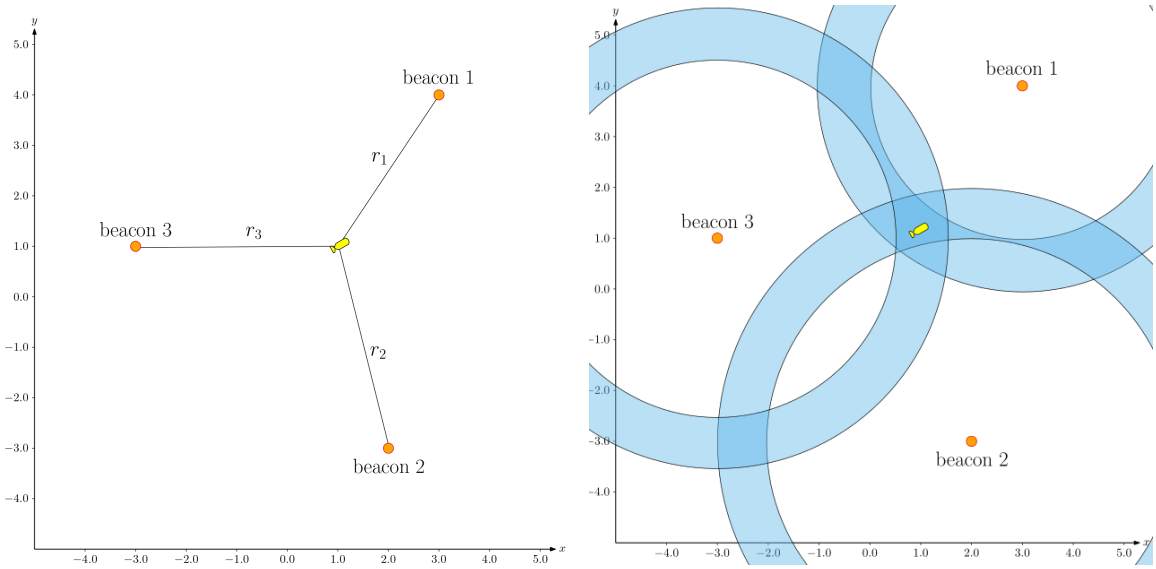
Remark 2.10. If $\mathbb{S}$ is found empty, *i.e* there exists no element $\mathbf{x}$ satisfying all constraints, then the CSP is *unsatisfiable*. In that case, one might be interested in the relaxed version of this CSP [38].

Example 2.9 (Localisation: CSP)**.** To illustrate the use of CSP, let us introduce a simple range-only localisation problem. Suppose that we have an AUV and 3 buoys as presented in Figure 2.5a. The AUV is equipped with acoustic sensors able to ping the buoys and receive their response with their position. The distances (r_1, r_2 and r_3) to the buoys can be computed providing that the clocks are synchronised. The sensors are not perfect so the measurements are enclosed in intervals $[r_1], [r_2], [r_3]$ (see Figure 2.5b). It is possible for the AUV to determine its position.

One can formalise this problem into a CSP:

- Let us denote x, y the position of the robot in the plane. We can set $\mathbf{x} = (x, y, r_1, r_2, r_3)$
- We now set the domain for $\mathbf{x}$ such that $[\mathbf{x}] = [-\infty, \infty] \times [-\infty, \infty] \times [r_1] \times [r_2] \times [r_3]$
- Lastly, given (x_1, y_1), (x_2, y_2), (x_3, y_3) the positions of the buoys, let us define the following constraints:

$$c_i : f_i(\mathbf{x}) = \sqrt{(x_i - x)^2 + (y_i - y)^2} - r_i = 0 \; i \in \{1, 2, 3\} \tag{2.22}$$



(a) Initial setup

(b) Initial setup with sensors measurements

Figure 2.5: Range-only problem using an AUV and three beacons

Sometimes computing the solution set is not achievable. Nevertheless, one might be able to compute an enclosure $[\mathbf{v}]$ of $\mathbb{S}$ such that $\mathbb{S} \subset [\mathbf{v}] \subset [\mathbf{x}]$. To do so, one could use tools called *contractors* coming with interval analysis to remove elements of $[\mathbf{x}]$ that do not satisfy the constraints of the CSP.

2.3.5 Contractors

In this section we present the tools used in interval analysis to contract CSPs. The following definition is extracted from [19].

Definition 2.8. A *contractor* associated with a constraint c_i of a CSP $\langle \sqcup$ with a solution set $\mathbb{S}$ is an operator $\mathcal{C}_i : \mathbb{IR}^n \to \mathbb{IR}^n$ such that:

1. $\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}_i([\mathbf{x}]) \subseteq [\mathbf{x}]$ (*contractance*)
2. $\begin{pmatrix} \mathcal{C}_i(\mathbf{x}) \\ \mathbf{x} \in [\mathbf{x}] \end{pmatrix} \Rightarrow \mathbf{x} \in \mathcal{C}_i([\mathbf{x}])$ (*consistency*)

Now let us detail what the different properties of the contractor imply:

- The contractance property ensures that no element outside of our input $[\mathbf{x}]$ can belong to the output.
- The consistency property guarantees the result, *i.e* if $\mathbf{x}$ satisfies the constraint c_i then it will not be removed from our enclosure of the solution set

Example 2.10 (Contractor)**.** Let us illustrate Definition 2.8 with a simple, graphic example. Let us introduce the set $\mathbb{S}$ painted yellow in Figure 2.6 and its associated contractor $\mathcal{C}_{\mathbb{S}}$. By associated contractor we mean the the constraint applied is

$$c : \mathbf{x} \in \mathbb{S}. \tag{2.23}$$

The input box $[\mathbf{x}]$ is painted black on the figure and the result after contraction $\mathcal{C}_{\mathbb{S}}([\mathbf{x}])$ is painted red. In this case the contractor is *minimal*, which means that the result of the contraction $\mathcal{C}_{\mathbb{S}}([\mathbf{x}])$ is the smallest box that contains $\mathbb{S} \cap [\mathbf{x}]$ without eliminating any solution [30].

Remark 2.11. This example also illustrates one of the limit of contractors which is the wrapping effect. Indeed, the set we are trying to characterise is rarely a perfect box which leads to an over-approximation that can be quite large, especially when the set is not convex. To overcome this obstacle one can use pavings which will be seen in Section 2.3.7

One of the main advantages of contractors is the fact that they can be combined together (union, intersection...) to get a better (smaller) approximation of the solution set.

Example 2.11 (Localisation: contractor)**.** Let us come back to our localisation problem introduced in Section 2.3.4. Let us see how contractors can be used to approximate our solution set. We introduce $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ the contractors associated with the distances measured to the three beacons. Figure 2.7 displays the results of $\mathcal{C}_1([\mathbf{x}]), \mathcal{C}_2([\mathbf{x}]), \mathcal{C}_3([\mathbf{x}])$ in red, blue and green respectively. The intersection is also displayed giving us an approximation of the true position of the robot. This enclosure could be even better by applying each contractor successively, until the result of the contraction stagnates , *i.e* for $i \in \{1, 2, 3\}, \mathcal{C}_i([\mathbf{x}]) = [\mathbf{x}]$. The result of this operation is called the fix point. An animation of this is available here[1]

[1] http://codac.io/tutorial/02-static-rangeonly/index.html

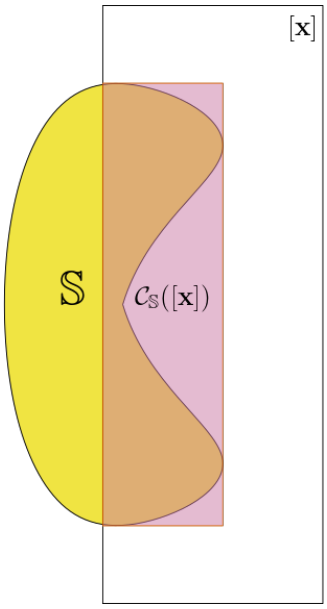
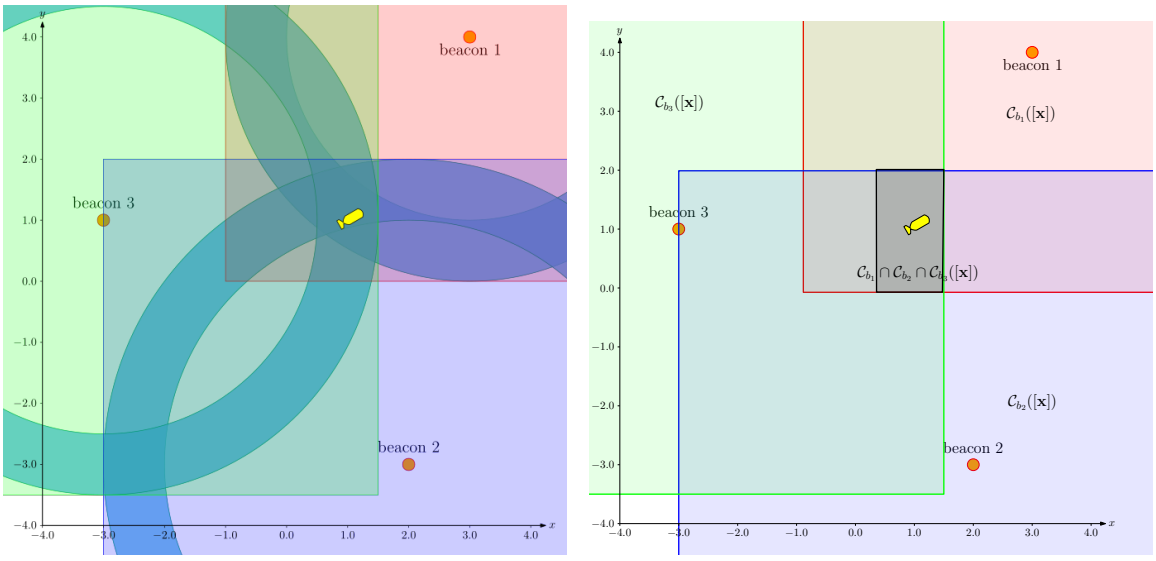


Figure 2.6: Example of contraction of a box $[\mathbf{x}]$ with a contractor $\mathcal{C}_{\mathbb{S}}$ over a set $\mathbb{S}$.



(a) Estimation for each measurement.

(b) Final estimation after intersection.

Figure 2.7: Contractors applied to the localisation problem.

2.3.6 Separators

When using contractors, we compute an over-approximation of our solution set. But we could also call it an outer approximation. The latter name could make one think of other methods from other mathematical fields where the enclosure of the solution is given by both an outer and an inner approximation such as numerical integration with rectangles. In our case that would mean starting from an empty set and adding new solutions to inflate our inner approximation. However, doing so would imply that we can easily and quickly compute our set $\mathbb{S}$ which is not the case. Instead, we are going to return the problem and look for the complementary set of $\mathbb{S}, \bar{\mathbb{S}}$. We will denote the contractor associated $\mathcal{C}_{\bar{\mathbb{S}}}$. From Section 2.3.5, this complementary contractor guarantees that the obtained result contains every non-solutions (and some solutions). The pair composed of the two contractors $\mathcal{C}_{\mathbb{S}}$ and $\mathcal{C}_{\bar{\mathbb{S}}}$ is called a *separator* [61].

This new operator $\mathcal{S}_{\mathbb{S}}$ takes as input a box $[\mathbf{x}]$ and yields two boxes $[\mathbf{x}_{in}]$, $[\mathbf{x}_{out}]$ such that:

$$[\mathbf{x}] \setminus [\mathbf{x}_{in}] \subset \mathbb{S} \tag{2.24a}$$

$$[\mathbf{x}] \setminus [\mathbf{x}_{out}] \cap \mathbb{S} = \emptyset \tag{2.24b}$$

Remark 2.12. The contractors $\mathcal{C}_{\mathbb{S}}$ and $\mathcal{C}_{\bar{\mathbb{S}}}$ are often denoted $\mathcal{C}_{out}$ and $\mathcal{C}_{in}$ as $\mathcal{C}_{\mathbb{S}}([\mathbf{x}]) = [\mathbf{x}_{out}]$ and $\mathcal{C}_{\bar{\mathbb{S}}}([\mathbf{x}]) = [\mathbf{x}_{in}]$.

Remark 2.13. Here, we are using the notations defined in [61]. It may seem counter-intuitive at first but one should keep in mind that instead of checking that solutions satisfy a constraint a contractor removes solutions that do not satisfy a constraint. Therefore the constraint is equivalent to "belongs to a set $\mathbb{S}$", a contractor $\mathcal{C}_{in}$ will remove elements that belongs to $\mathbb{S}$. Hence the result given in Equation (2.24a). In the same manner a contractor $\mathcal{C}_{out}$ will remove elements that do not belong to $\mathbb{S}$. This is illustrated in Figure 2.8. This notation is also the one used to implement separators in Interval Based Explorer (IBEX), that is why we will keep it to help the reader understand pieces of codes presented later in this work.

Example 2.12 (Separator)**.** We introduce the set $\mathbb{S}$ and a box $[\mathbf{x}]$ as presented in Figure 2.8. The application of the separator $\mathcal{S}_{\mathbb{S}}$ on $[\mathbf{x}]$ yields two boxes $[\mathbf{x}_{in}]$, $[\mathbf{x}_{out}]$ painted blue and pink respectively. As it is shown, $[\mathbf{x}_{in}] \cap [\mathbf{x}_{out}] \neq \emptyset$. This area is area is called the *remainder* [61].

2.3.7 Pavings

As mentioned in Section 2.3.5, to limit the pessimism induced by the wrapping effect, one may use a paving algorithm to increase the accuracy of the result obtained with contractors or separators. This method consists in dividing the search space into non overlapping smaller boxes to increase the finesse of the result. The set gathering these boxes is called a *paving*. This allows better enclosure of complex sets. The paving method is often used when trying to perform set inversion (see Section 2.3.8). An example of paving method coupled with set inversion is given in Figure 2.9.

Remark 2.14. This technique should be used as a last resort because of the complexity of the paving algorithm. Indeed adding a new dimensions of the problem means adding a new

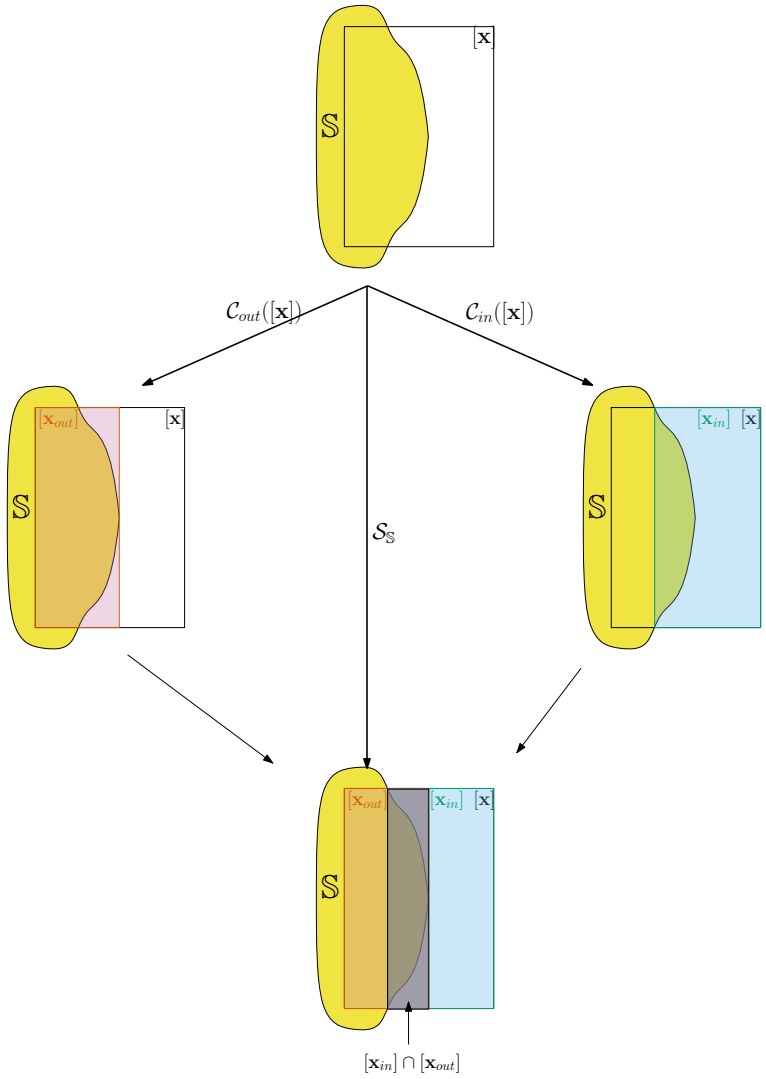


Figure 2.8: Application of a separator on a box

dimension to pave. Therefore the number of boxes to be considered increases, leading to a greater need in computational power and processing time. Hence limiting the efficiency of the method.

2.3.8 Set inversion

In numerous cases, especially in robotics, the sets we would like to characterise are not observable, *i.e* we cannot measure them directly. Nevertheless, an observation of the images of these sets is available. Mathematically speaking, this means that we are looking for a set $\mathbb{X}$ which is the preimage of a set $\mathbb{Y}$ by a function f such that $f(\mathbb{X}) = \mathbb{Y}$. Thus the solution set we are looking for is $\mathbb{S} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\}$. A CSP is a set inversion problem by definition. Solving a CSP comes down to characterizing a set $\mathbb{S}$ such that its image by a function f belongs to a set of constraints $\mathbb{Y}$. Thus interval analysis is well equipped to

solve this kind of problem even in non linear situations [85].

We can illustrate this with Example 2.9. The set we want to characterise is the set of possible positions $\mathbb{X} \subset \mathbb{R}^2$ of our robot. However, the only information we have are the range measurements, defining a set $\mathbb{Y} \subset \mathbb{R}^3$. We have a function $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^3$ (see Equation (2.22)), linking $\mathbb{X}$ to $\mathbb{Y}$. It is thus possible to compute the set $\mathbb{S}$ of positions from which the measurements could have been made.

An approximation of this set can be computed using the Set Inversion *via* Interval Analysis (SIVIA) algorithm [62]. This algorithm combines both paving method and set inversion. It takes as input the search space and returns three distinct lists of boxes (or subpavings). The boxes belonging to the outer approximation, the ones in the inner approximation and a list of indeterminate boxes, on which it cannot assert whether they belong to one or the other approximation. The algorithm works as follows:

1. It takes a box $[\mathbf{x}]$ as input
2. It computes the image $[\mathbf{y}]$ of $[\mathbf{x}]$ by $\mathbf{f}$
3. It checks whether this image belongs to our constraint set $\mathbb{Y}$
4. Depending on the results it can be placed either in the inner or outer list or considered uncertain.
5. If the box is noted as uncertain it checks if its maximum width is inferior to a parameter ϵ set by the user
6. If it is not, the box is bisected along its wider dimension and the same process is applied from step 1 to the two resulting sub-boxes. Otherwise the box is placed in the uncertain boxes list

The formalised algorithm is given in [62]. Naturally the smaller the parameter ϵ, the more accurate the result will be however it will require a longer processing time.

Example 2.13 (Localisation: set inversion)**.** The result of the SIVIA algorithm applied to our localisation problem is given in Figure 2.9

Remark 2.15. In a guaranteed context, one should consider both the inner approximation and the boundary if the result is meant to be used as proof.

Remark 2.16. Computing an inner and an outer approximation is of primary importance for certain situations such as reachability analysis [72] or viability problems [5]. One may also use these to design or validate controllers for dynamical system [79].

Remark 2.17. For all interval computations made throughout this thesis using intervals, we will use the IBEX library [18]. This library implements generic contractors and separators and also allows us to work with inclusion functions. It is based on the low-level library GAOL [46] which handles the floating point computations and basic operators.

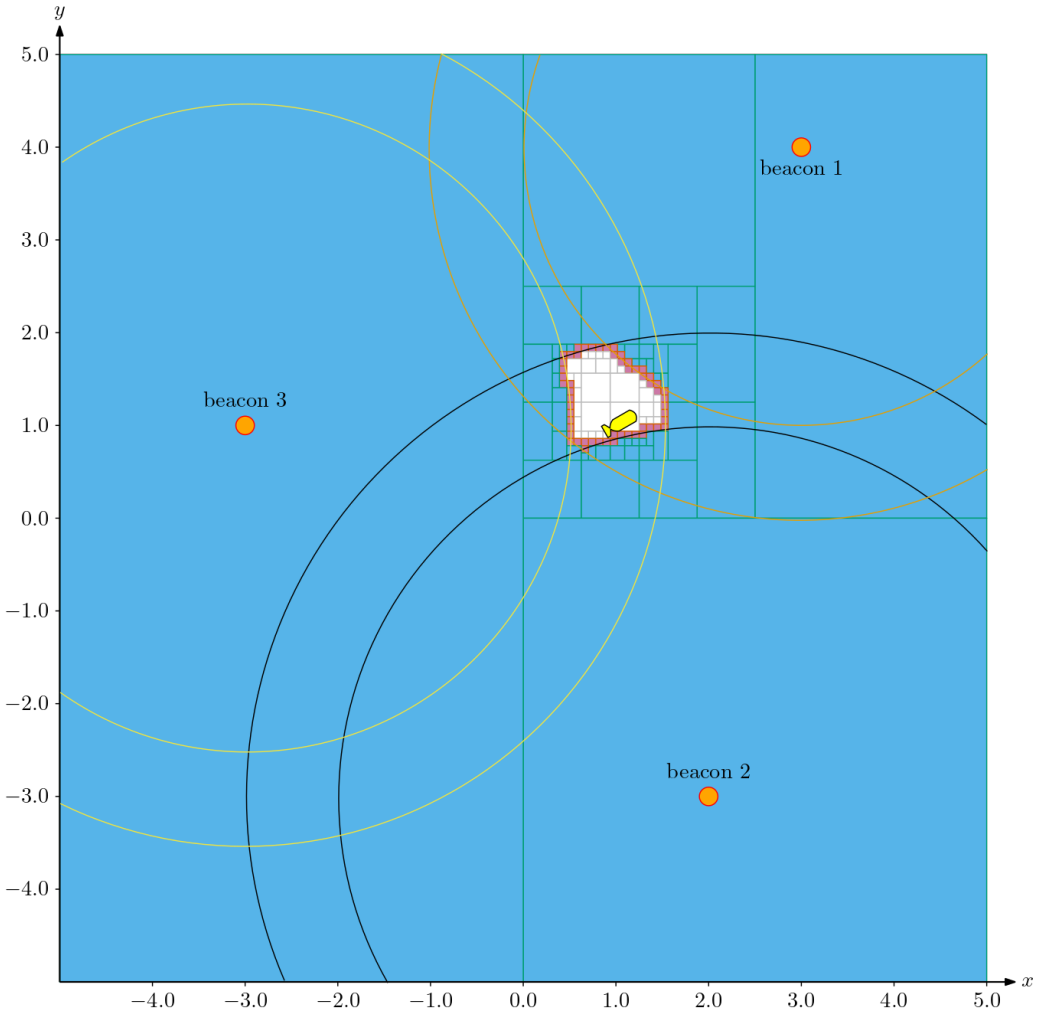


Figure 2.9: SIVIA algorithm applied to the localisation problem

2.4 Tubes

In the localisation example developed from Section 2.3.4 to Section 2.3.8 the situation analysed is a static one. However, when it comes to mobile robotics, the whole trajectory of the robot is of greater interest than a sole position at a defined time t. Moreover, the different range measurements made by the sensors rarely come in a synchronised manner and the system evolves between each new acquisition. Hence the need for a tool that can deal with all these constraints even in non-linear cases and with strong uncertainties.

Example 2.14 (Localisation: trajectory)**.** Let us extend the static localisation example. We will now the consider the following state estimation problem represented by the IVP:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) & \text{(2.25a)} \\ r_i = g(\mathbf{x}(t_i)) \ \forall i \in \{1, 2, 3\} & \text{(2.25b)} \end{cases}$$

We now have a new constraint which is the evolution constraint given by Equation (2.25a),

represented by a differential equation as seen in Section 2.2, in addition to the measurements constraints represented by Equation (2.25b). The careful reader will have noticed that the measurements are also asynchronous in this case. This problem is depicted in Figure 2.10.

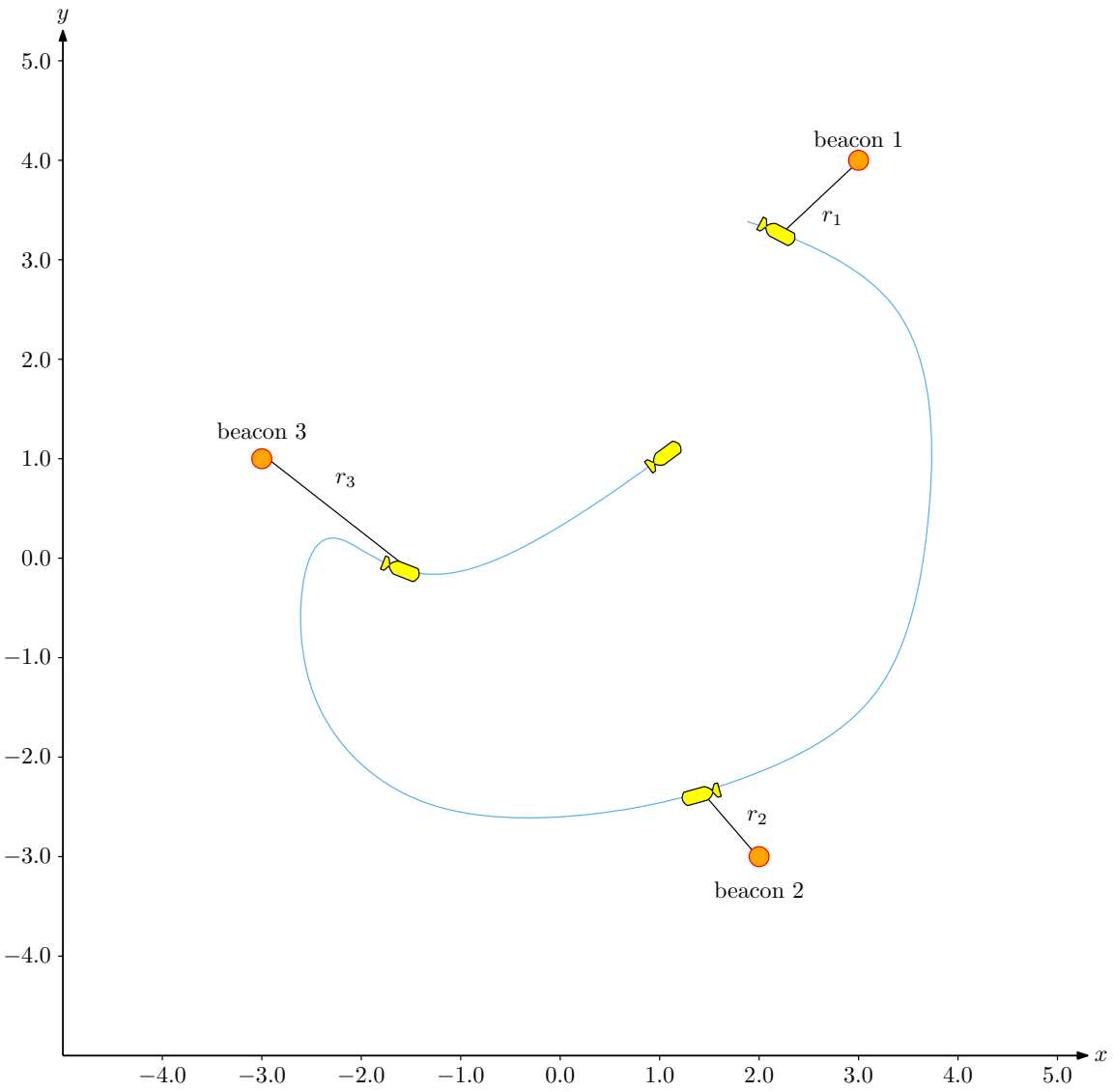


Figure 2.10: Range only localisation problem in a dynamical context. The trajectory of the robot is depicted in blue. The three black lines show the range measurements made at t_1, t_2 and t_3

To handle such problem, the concept of constraint programming has been extended to differential constraints. The subject has first been studied in [78] and [9] before being explored thoroughly in [103]. In this section we will define and introduce the different notations used in this field derived from [103, Chapter 2].

2.4.1 Definitions

We introduce the notion of *trajectory* which is a single variable function depending only on the independent evolution variable t. The image of t is the trajectory value representing a system state, an observation and so on. Mathematically, a trajectory can be written as: $x(\cdot) : \mathbb{R} \to \mathbb{R}$. Then $x(t)$ is the *evaluation* of the trajectory $x(\cdot)$ at time $t \in \mathbb{R}$.

The concept of tube first appeared in the field of ellipsoidal estimation in [71, 35]. It can be defined as an envelope of trajectories $x(\cdot) : \mathbb{R} \to \mathbb{R}$ defined over a time domain $[t_0, t_f]$. It is called an envelope as there may exist trajectories enclosed in the tube that are not solutions of the problem considered. The notations used in the latter part of this work will be the ones introduced in [78, 9] where a tube is denoted $[x](\cdot) : \mathbb{R} \to \mathbb{IR}$. It is an interval of two trajectories $[x^-, x^+]$ such that $\forall t \in [t_0, t_f], x^-(t) \leq x^+(t)$. In the same manner as the empty set, the empty tube is denoted by $\emptyset(\cdot)$. A trajectory $x(\cdot)$ is considered enclosed in the tube $[x](\cdot)$ if $\forall t \in [t_0, t_f],\ x^-(t) \leq x(t) \leq x^+(t)$.

Remark 2.18. We often represent the tube only on what is called its *time domain* $[t_0, t_f] \subset \mathbb{R}$(see Figure 2.11). However, the tube is defined on all $\mathbb{R}$. As shown on Figure 2.12, $\forall t \in \mathbb{R}[\backslash][t_0, t_f], [x](t) = [-\infty, \infty]$.

Remark 2.19. Trajectories and tubes can be stacked in the same manner intervals are. In that case, we speak about trajectory vectors and tube vector denoted $\mathbf{x}(\cdot) : \mathbb{R} \to \mathbb{R}^n$ and $[\mathbf{x}(\cdot)] : \mathbb{R} \to \mathbb{IR}^n$ respectively.

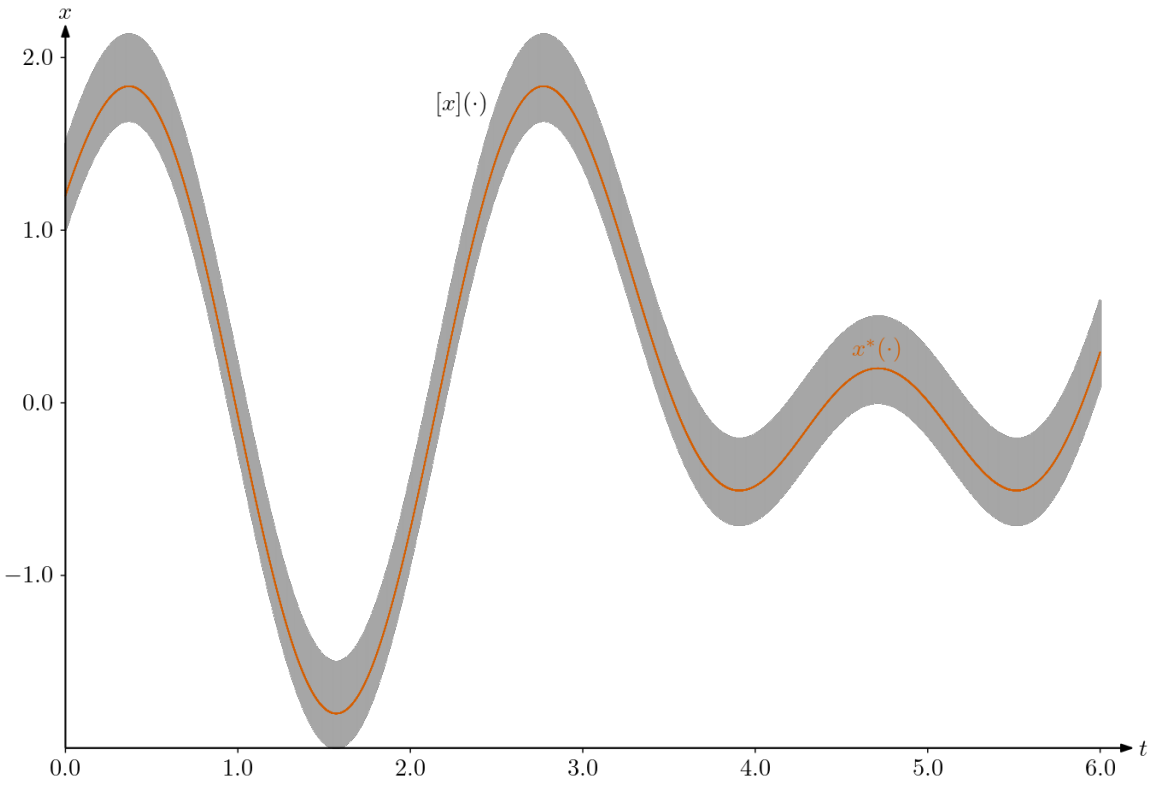


Figure 2.11: A one dimensional tube $[x](\cdot)$ enclosing a trajectory $x^*(\cdot), \forall t \in [t_0 = 0, t_f = 6]$.

2.4.2 Implementation of tubes

Mathematically, tubes are continuous objects w.r.t time. However as mentioned at the beginning of Section 2.3, a computer cannot represent all numbers of $\mathbb{R}$. Therefore choices have been made to represent such an object in a computer. For all computations with tubes, we will use Catalog Of Domains And Contractors (Codac) [102]. In this library, tubes are represented as a sequence of slices as shown in Figure 2.12. Consider a real n-dimensional tube $[\mathbf{x}](\cdot)$. In its computer representation, each slice over a time domain $[k\delta, k\delta + \delta]$ is a box enclosing the evaluation of $[x](\cdot)$ on this time domain. Thus $[\mathbf{x}](t)_{computer}$ is constant for $t \in [k\delta, k\delta + \delta]$ is constant on each slice. The width δ is usually fixed but tubes with slices of different width are available.

We give below a short piece of C++ code (see Listing 2.1) which creates the tube and trajectory presented in Figure 2.12. Throughout this thesis, we will provide chunks of C++ code for the reader to test the library by himself/herself. It has to be noted that Codac is also available in Python. One can find all needed information for installation here[2]

Listing 2.1 Implementing a tube and a trajectory with Codac V1 in C++.

```
#include "codac.h" // import the library

using namespace codac;

int main()
{
    // Time domain on which the tube is defined
    Interval time_domain(0,6);
    // Time step used to create the tube or trajectory
    double delta = 0.1;

    // analytic expression of the tube [x](.)_computer
    TFunction f("((sin(3*t)+cos(2*t))+[0,0.5])");
    //Creating the tube
    Tube x(time_domain,delta,f);

    // analytic expression of the trajectory x*(.)
    TFunction f_traj("((sin(3*t)+cos(2*t))+0.2)");
    //Creating the trajectory
    Trajectory x_star(time_domain,f_traj,delta);

    // Graphic part
    // ...

}
```

[2] http://codac.io/install/01-installation.html

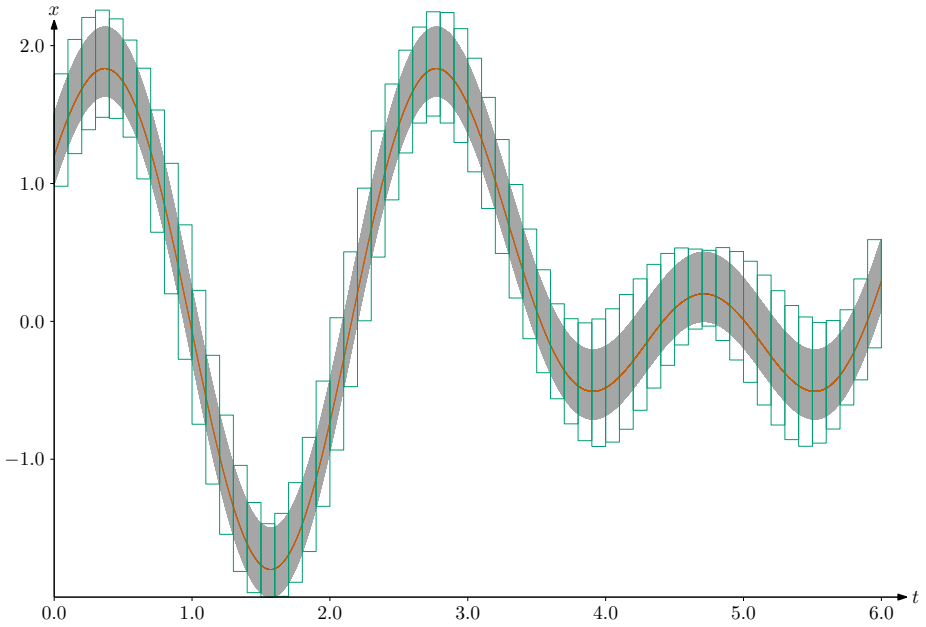


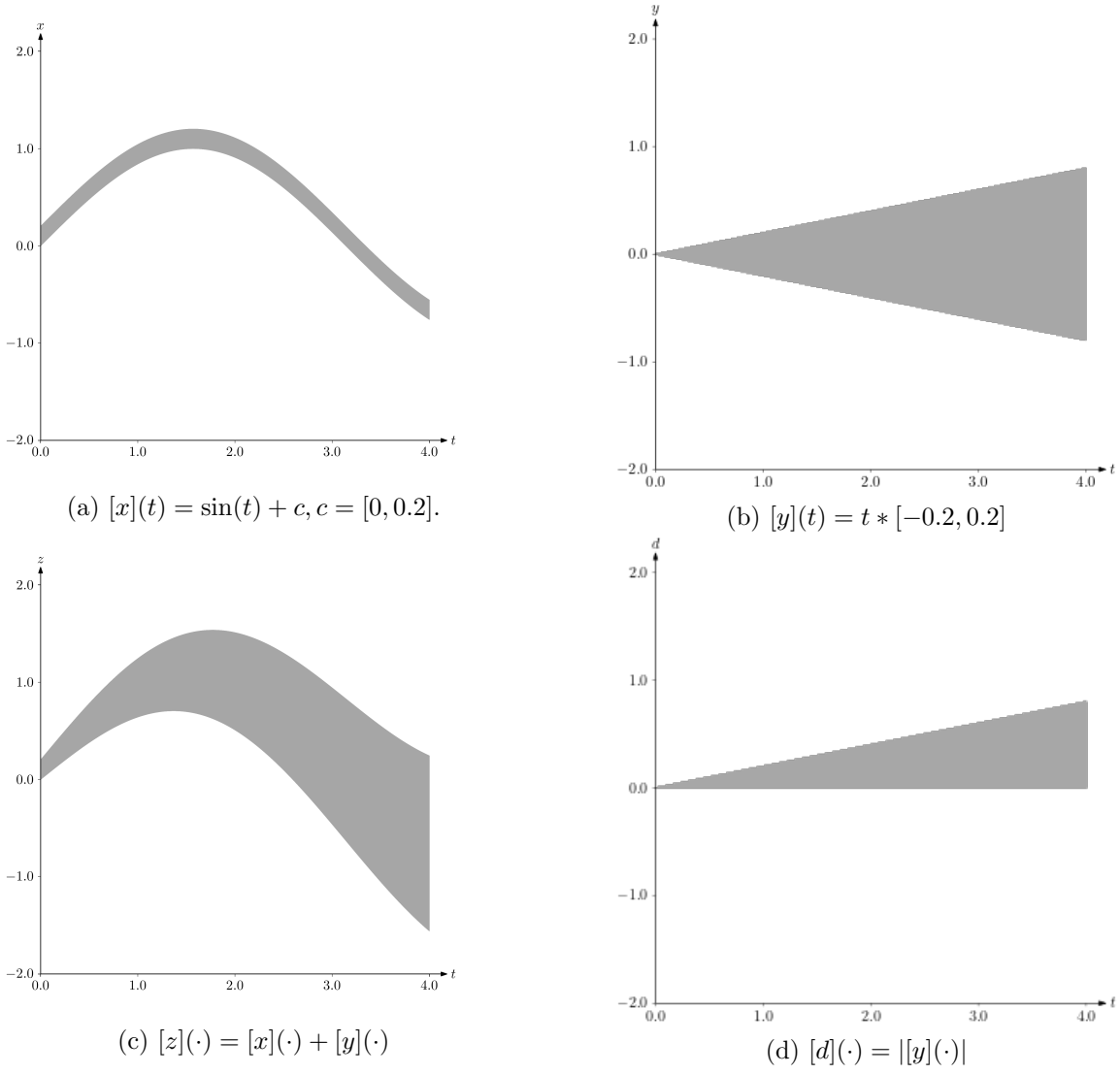
Figure 2.12: Representation of a tube $[\mathbf{x}](\cdot)$ in Codac. The two grey boxes at the endpoints of the time domain represent the value of the tube such that $\forall t \in [-\infty, t_0[\cup]t_f, \infty], [x](t) = \mathbb{IR}$.

2.4.3 Operators

Numerous operators can be applied to our newly introduced tube object. They can be divided into two categories: operators extended from interval analysis and operators dedicated to tubes as they are linked to the dynamical aspect of them.

2.4.3.1 Operators extended from interval analysis

The first category of operators gathers all the operators extended form interval analysis. Hence the classical binary operators can be applied along with the union and intersection. Contractors and inclusion functions are also operators applicable. Unlike regular intervals, separators cannot be developed for tubes. Some examples of operators extended from interval analysis are available in Figure 2.13. The code to generate these tubes is also given in Listing 2.2.



(a) $[x](t) = \sin(t) + c, c = [0, 0.2]$.

(b) $[y](t) = t * [-0.2, 0.2]$

(c) $[z](\cdot) = [x](\cdot) + [y](\cdot)$

(d) $[d](\cdot) = |[y](\cdot)|$

Figure 2.13: Examples of basic operators applied on tubes

2.4.3.2 Temporal operators

This second category is composed of two types of operators

- evaluations (interval evaluation and tube inversion),
- temporal contractors.

Let us begin with the evaluation operators.

Definition 2.9 (tube evaluation)**.** The interval evaluation of a tube $[x](\cdot)$ over a bounded

Listing 2.2 Implementation of basic operators

```
// Define time domain, time step ...

// Creating the tube [x](.)
TFunction fx("((sin(t)+[0,0.2])");
Tube x(time_domain,delta,fx);
// Creating the tube [y](.)
TFunction fy("(t*[-0.2,0.2])");
Tube y(time_domain,delta,fy);
//Computing [z](.)
Tube z = x+y;
// Computing [d](.)
Tube d = abs(y);
```

domain $[t]$ is given in [10]:

$$[x]([t]) = [\{x(t)|x(.) \in [x](t), t \in [t]\}] \tag{2.26}$$

$$= \bigsqcup_{t\in[t]} [x](t) \tag{2.27}$$

where $[x](t)$ is the smallest box enclosing all solutions for $x(t)$ such that $x(\cdot) \in [x](\cdot), t \in [t]$. An example of evaluation is given in Figure 2.14.

Definition 2.10 (tube inversion)**.** The tube inversion, denoted by $[x]^{-1}([y])$, is defined by:

$$[x]^{-1}([y]) = \bigsqcup_{y\in[y]} \{t|y \in [x](t)\} \tag{2.28}$$

As shown in Figure 2.15, the resulting interval encloses all the preimages of $[y]$ by $[x](\cdot)$.

Let us move on to the last operators we will present here, the tube-dedicated contractors. We will focus on two of them, the one associated with the derivative constraint and the one dedicated to the evaluation constraint presented in [103].

Differential contractor for the constraint $c_{\frac{d}{dt}} : \dot{x}(\cdot) = v(\cdot)$

Consider two trajectories, $x(\cdot)$ and $v(\cdot)$, approximated by the tubes $[x](\cdot)$ and $[v](\cdot)$ respectively on a domain $[t_0, t_f]$. These two trajectories are linked through the constraint $c_{\frac{d}{dt}} : \dot{x}(\cdot) = v(\cdot)$. We would like to reduce these tubes using this differential constraint in a guaranteed way, *i.e* without loosing any trajectory enclosed in $[x](\cdot)$ and $[v](\cdot)$ that satisfies the constraint. A dedicated contractor, denoted $\mathcal{C}_{\frac{d}{dt}}$ has been developed to contract the tubes w.r.t this constraint [103, Chapter 3].

Proposition 2.3. *The operator* $\mathcal{C}_{\frac{d}{dt}}$ *is a contractor for the constraint* $c_{\frac{d}{dt}}$ *and is defined by:*

$$\begin{pmatrix} [x](t) \\ [v](t) \end{pmatrix} \overset{\mathcal{C}_{\frac{d}{dt}}}{\longmapsto} \begin{pmatrix} \bigcap_{t_1=t_0}^{t_f} \left([x](t_1) + \int_{t_1}^{t} [v](\tau)d\tau\right) \\ [v](t) \end{pmatrix}, \tag{2.29}$$

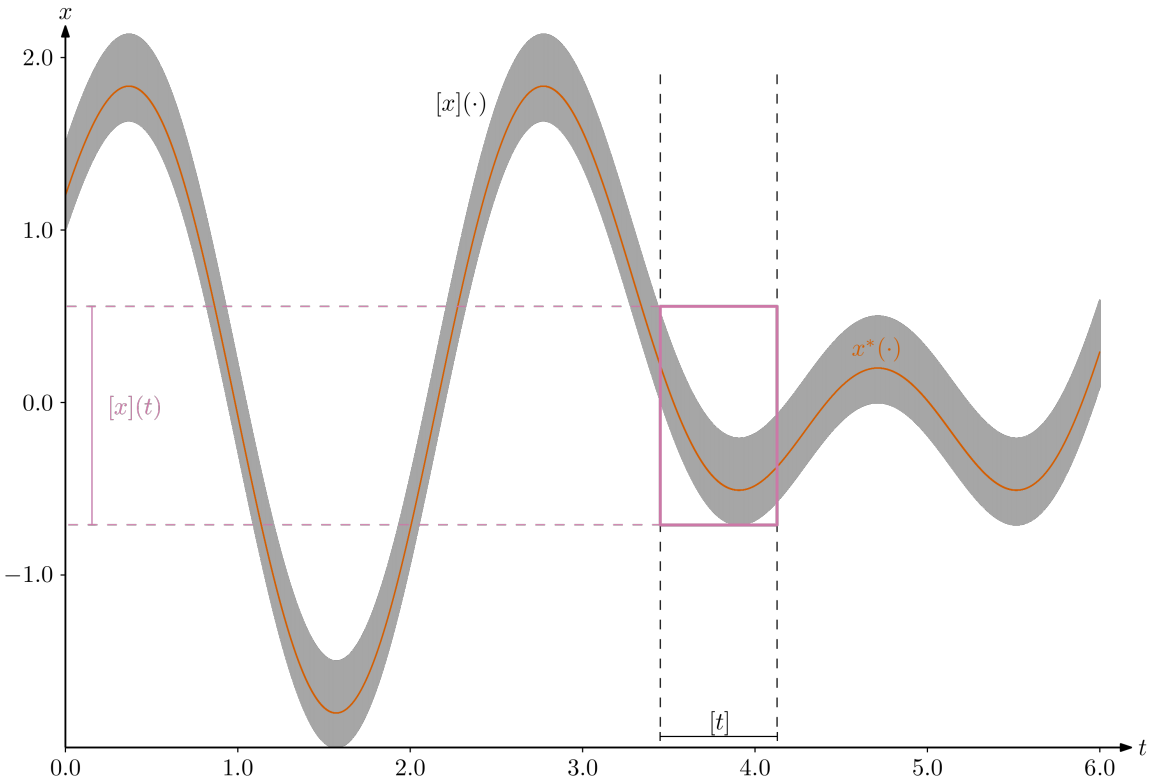


Figure 2.14: Example of the evaluation of $[x](\cdot)$ on the time interval $[t]$

where $[t_0, t_f]$ *is the definition domain of both* $[x](\cdot)$ *and* $[v](\cdot)$.

To be a contractor, the operator $\mathcal{C}_{\frac{d}{dt}}$ defined above must satisfy both the contractance and consistency properties seen in Section 2.3.5. The proof can be found in [103, Chapter 3].

Example 2.15 ($\mathcal{C}_{\frac{d}{dt}}$)**.** Consider the tubes $[x](\cdot), [v](\cdot)$ such that:

$$[x](t) = t * [-0.5, 0.5],$$
$$[v](t) = [0.2, 0.3],$$

for $t \in [0, 10]$ and $[\dot{x}](\cdot) = [v](\cdot)$ We apply the $\mathcal{C}_{\frac{d}{dt}}$ to $[x](\cdot)$ and $[v](\cdot)$. The result is shown in Figure 2.16. The piece code uses to generate this tubes and apply the contractor is also given in Listing 2.3

Remark 2.20. As one can see on Figure 2.16, only the tube $[x](\cdot)$ has been contracted. This was anticipated from the contractor definition in Equation (2.29). Indeed, it is not possible to bound the evolution of a trajectory enclosed in the tube $[x](\cdot)$, unless the tube is degenerate *i.e* it encloses only one trajectory. Thus the derivative $\dot{x}(\cdot) \in [v](\cdot)$ can take any arbitrary value. Therefore we cannot propagate information from $[x](\cdot)$ to $[v](\cdot)$ as it is possible from $[v](\cdot)$ to $[x](\cdot)$. An example of such a trajectory $x^*(\cdot)$ has been drawn on Figure 2.16 to illustrate this.

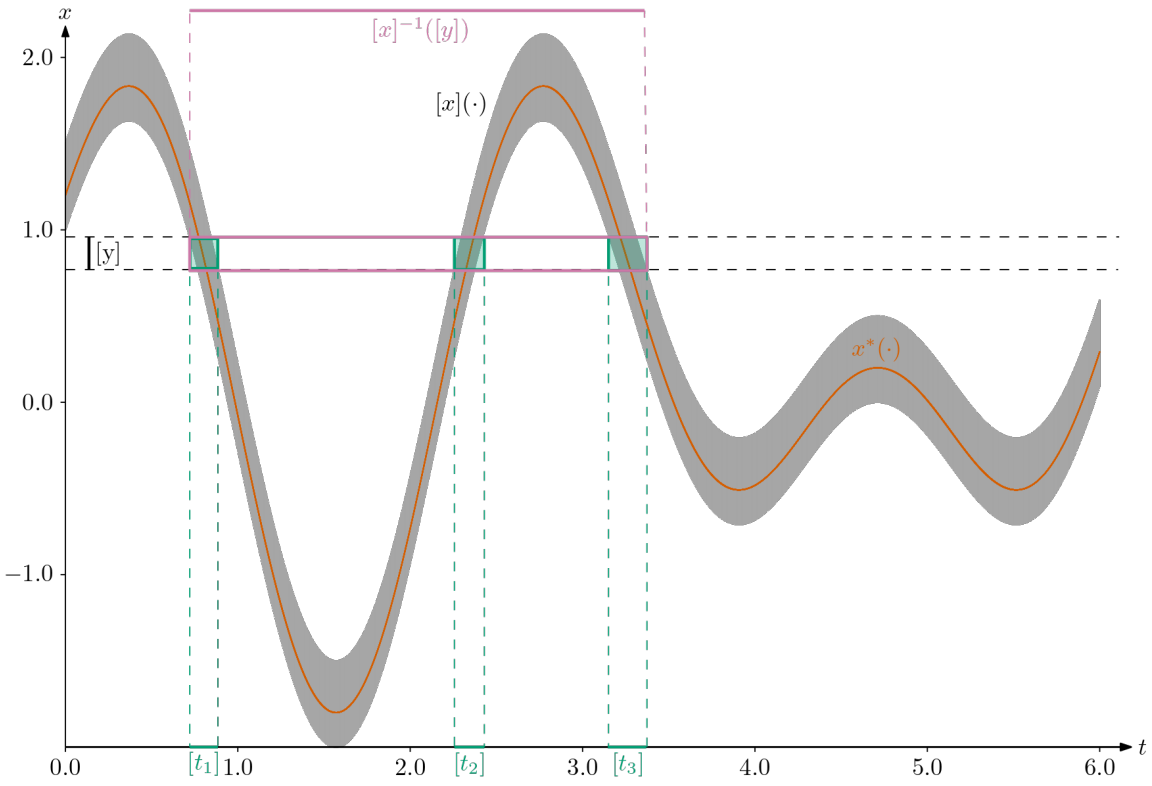


Figure 2.15: Example of the inversion of $[y]$ by the tube $[x](\cdot)$

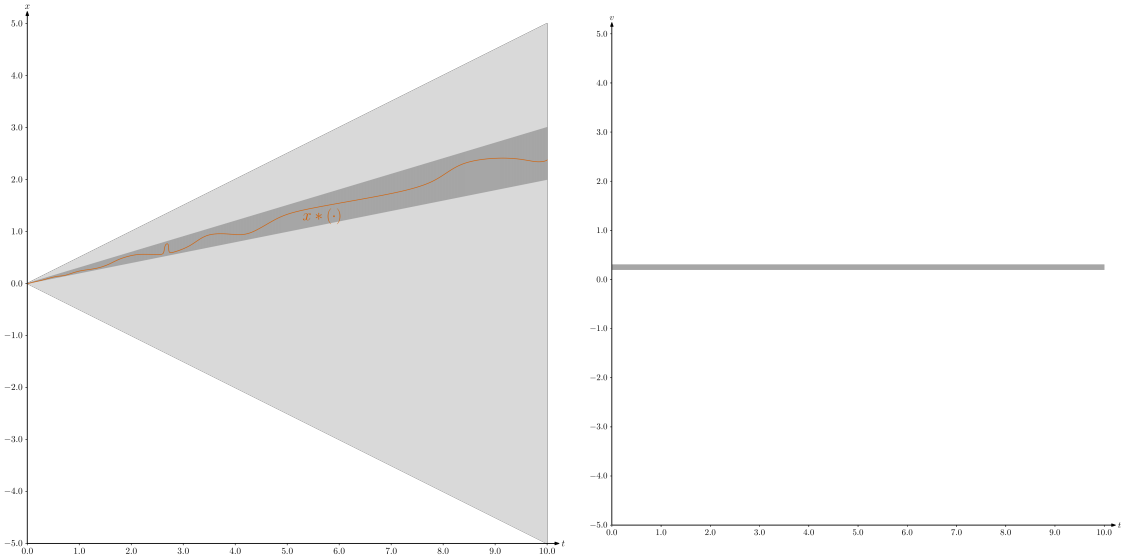
This contractor is really convenient in a robotics context. Indeed, as we have seen in Section 2.2, we often model a dynamical system using its evolution function of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ as we have more information on the way it evolves, *i.e* its derivatives, than its actual trajectory. Thus, a tube enclosing an approximation of the trajectory of a robot, obtained through measurements, can be contracted using this derivative, reducing its envelope and giving us a more accurate idea of the real trajectory.

Measurement contractor for the constraint $c_{\text{eval}} : z = x(t)$

Consider a tube $[x](\cdot)$ enclosing a trajectory $x(\cdot)$ over time domain $[t_0, t_f]$. Consider also an interval $[z]$ which encloses an observation z of $x(\cdot)$ at time t. Again, t is also enclosed in $[t]$. We would like to contract $[x](\cdot)$, $[z]$ and $[t]$ knowing that $z = x(t)$. Hence, the set we are trying to characterise is the following one:

$$\mathbb{S}_{\text{eval}} : (t, z, x(.)), t \in [t], z \in [z], x(\cdot) \in [x](\cdot) | z = x(t) \tag{2.30}$$

The contractor we need will aim at intersecting the tube $[x](\cdot)$ with the minimal tube enclosing all the trajectories that satisfy the measurement *i.e* that go through the box $[t] \times [z]$ as shown in Figure 2.17. One should keep in mind that a compliant trajectory does not need to be contained in $[z]$ over the whole time interval $[t]$. Thus the contractor must take into account the evolution of the trajectory. To achieve this, the operator will require the knowledge of the derivative $\dot{x}(\cdot) = v$ of $x(\cdot)$. A generic contractor called $\mathcal{C}_{\text{eval}}$ as been defined in [103] for such an application.



(a) $[x](\cdot)$ (b) $[v](\cdot)$

Figure 2.16: $\mathcal{C}_{\frac{d}{dt}}$ contractor applied to $[x](\cdot)$ and $[v](\cdot)$ knowing that $\dot{x}(\cdot) = v(\cdot)$. The tubes painted in light grey are the tubes before contraction and the ones in darker grey are the one after contraction.

Proposition 2.4. *A contractor $\mathcal{C}_{eval}([t], [z], [x](\cdot), [v](\cdot))$ appliying c_{eval} on intervals and tubes is defined by:*

$$\begin{pmatrix} [t] \\ [z] \\ [x](\cdot) \\ [v](\cdot) \end{pmatrix} \xrightarrow{\mathcal{C}_{eval}} \begin{pmatrix} [t] \cap [x]^{-1}([z]) \\ [z] \cap [x]([t]) \\ [x](.) \cap \bigsqcup_{t_1 \in [t]} \left(([x](t_1) \cap [z]) + \int_{t_1}^{\cdot} [\dot{x}](\tau) d\tau \right) \\ [\dot{x}](t) \end{pmatrix} \tag{2.31}$$

Example 2.16. Consider a tube $[x](\cdot)$ representing the evolution of the yaw of a vehicle given by a model. We also have it derivative given represented by $[v](\cdot)$

$$[x](t) = \sin(t) + [-0.5, 0.5]$$
$$[v](t) = \cos(t)$$

At some time $t_1 \in t = [6, 7]$, an observation is made using a compass. This measurement is enclosed in the interval $[z] = [0.8, 1.5]$. The careful reader may have noticed that the exact moment of observation is not known. Let us apply the contractor $\mathcal{C}_{\text{eval}}$ on $[x](\cdot), [v](\cdot), [z]$ and $[t]$. The piece of code to perform these operation is given in Listing 2.4.

One can notice that $[x](\cdot), [z]$ and $[t]$ have been contracted. Hence giving us a better approximation of the time of the observation, the observation itself, and the yaw computed by the model.

Again this contractor fits perfectly in a robotics context. For instance, in a localisation problem, we may have an idea of the trajectory of the vehicle. We represent it as a tube

Listing 2.3 Using the contractor $\mathcal{C}_{\frac{d}{dt}}$.

```
// Define time domain, time step ...

TFunction fx("t*[-0.5,0.5]");
TFunction fv("[0.2,0.3]");
Tube x(time_domain,time_step,fx);
Tube v(time_domain,time_step,fv);
// Creating the derivative contractor
CtcDeriv c_deriv;
// Applying the derivative contractor to [x](.) and [v](.)
c_deriv.contract(x,v);
```

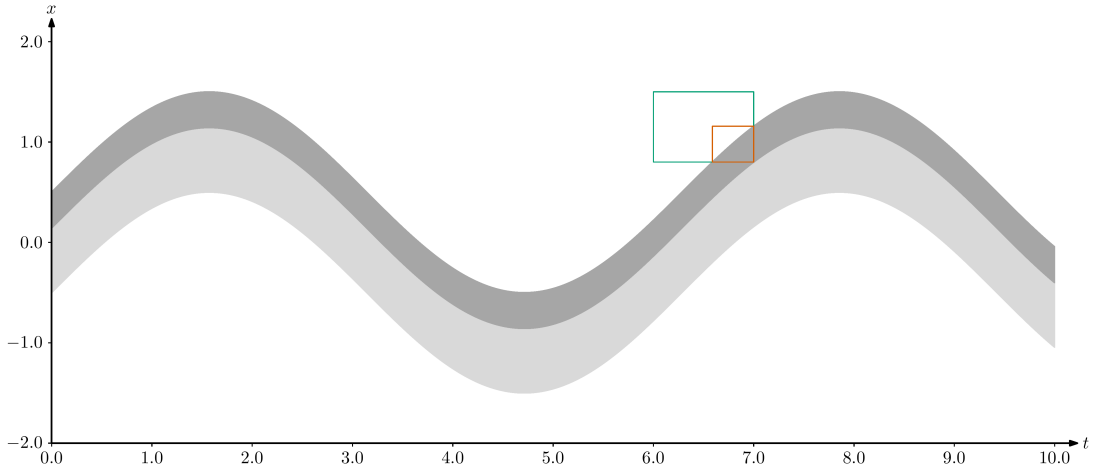


Figure 2.17: $\mathcal{C}_{\text{eval}}$ contractor applied to $[x](\cdot)$,$[z]$ and $[t]$

enclosing the true trajectory with some uncertainty. At some time t, an observation z is made by one of the sensors the robot is equipped with. As both the sensor and the clock are not perfect t and z are enclosed in $[t]$ and $[z]$ respectively. This problem will be detailed and treated in Chapter 6.

Listing 2.4 Using the contractor $\mathcal{C}_{\text{eval}}$.

```
// Define time domain, time step ...

TFunction fx("sin(t)+[-0.5,0.5]");
TFunction fv("cos(t)");
Tube x(time_domain,time_step,fx);
Tube v(time_domain,time_step,fv);

Interval t(6,7);
Interval z(0.8,1.5);
CtcEval c_eval;

c_eval.contract(t,z,x,v);
```

CHAPTER 3

INTRODUCTION TO GUARANTEED INTEGRATION

Contents

3.1 Introduction

In the previous sections (Section 2.2, Section 2.3, Section 2.4), we stumbled upon the equation $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ or its extended version $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$. We mentioned in Section 2.2.1, that using this equation and an initial condition in a numerical integration scheme, one could compute the trajectory of a robot. Nonetheless, the conventional Euler integration scheme or the bit more complex Runge-Kutta scheme were designed to work with a precise initial condition (a defined point) without any uncertainties. In our case we would like our integration scheme to take into account uncertainties on both the initial condition but also system parameters. In Section 2.3 we presented the tool used to handle uncertainties, *i.e.* interval analysis. In this section we will give brief review of algorithms developed to perform a guaranteed numerical integration, taking these uncertainties into account, *i.e* such that all feasible solutions are enclosed in a reliable set. We will roughly explain their method as we will use one of them but especially underline their main drawbacks, which were the motivation of the work presented in Section 3.4.5. For a far more exhaustive review presentation, the reader may refer to [14].

3.2 Origins

Currently, the issue of computing a guaranteed envelope of the integral of a function is nothing new. Krückeberg and Moore already addressed it in the 60's [69, 89, 90]. They both presented methods to solve an IVP in a guaranteed way. However the main drawback of them was, again when it comes to interval analysis, the wrapping effect induced when working with axis-aligned boxes. This leads to a bloating phenomenon after a few steps of computation. Since then, most algorithms developed focused on tackling this issue using various methods, to give a better (smaller) enclosure of the solution.

3.3 The basic method

Let us come back to Equation (3.1) that motivated this research with an arbitrary initial condition $\mathbf{x}_0$

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)). \tag{3.1}$$

From here, we added that our initial condition was rarely precisely known but uncertain. Therefore, we will encapsulate our uncertain initial state in a box $[\mathbf{x}_0]$. Moreover, our evolution function $\mathbf{f}$ will probably depend on the system characteristics and its interactions with its environment. For instance, its mass, frictions, currents and so on ... These different parameters are rarely known over time but they can be bounded. Hence, it is possible to represent them as intervals. With all this we can transform our original *differential equation* 3.1 into a *differential inclusion* given in Equation (3.2)

$$\dot{\mathbf{x}} \in [\mathbf{f}]([\mathbf{x}]). \tag{3.2}$$

Remark 3.1. The tools used later in this thesis can work with differential inclusions with the limit that the uncertain parameters are constant over the time of integration. This is due to the fact that they need to differentiate $\mathbf{f}$ with respect to time. Therefore, if the uncertain parameters are not constant, these methods would need an enclosure of their derivatives during computation. Sadly, it is hardly likely that even one of them is available in a robotics context. Hence we will limit ourselves to work with differential inclusions that contain, at most, constant uncertain parameters.

Example 3.1 (localisation: dynamical context)**.** Once again we will use our localisation example. This time, instead on focusing on the distance constraints relatively to the buoys, we will focus on the evolution constraint, introduced in our extended Example 2.14. The state $\mathbf{x}$ of our vehicle is now represented by a tripletons such that $\mathbf{x} = (x, y, \theta)^T$. Its evolution is modelled by the following equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} v_p \cdot \cos(\theta) \\ v_p \cdot \sin(\theta) \\ \omega_p \end{pmatrix}, \tag{3.3}$$

where v_p and ω_p are constant tangential and rotational speed parameters enclosed in intervals. The naive solution would be to integrate our differential equation using an Euler scheme with the inclusion function counterpart of $\mathbf{f}$, $[\mathbf{f}]$, which means solving the equation below in a numerical way.

$$[\mathbf{x}](t + dt) = [\mathbf{x}](t) + \int_t^{t+dt} [\mathbf{f}]([\mathbf{x}](\tau))d\tau \tag{3.4}$$

Then applying an Euler scheme to Equation (3.4) would yield:

$$[\mathbf{x}](t + dt) = [\mathbf{x}](t) + [0, dt][\mathbf{f}]([\mathbf{x}]([t, t + dt])) \tag{3.5}$$

Hence we would need a box $[\mathbf{x}]([t, t + dt])$ such that its encloses all the trajectories coming from $[\mathbf{x}](t)$ and evolving during dt. This box $[\chi]$ is called the *global enclosure* and its is not directly available as there is no available formula but is computed through an iterative process.

However, the Euler scheme is quickly limited and one might need a better one to obtain acceptable results. Thus better guaranteed integration algorithms were developed. Most can be classified into one of the two following groups:

1. Algorithms based on the Taylor expansion method (see [34, 39, 69, 82, 88, 120, 124])
2. Algorithms based on the Hermite-Obreshkov expansion method [93]

In this thesis, we will use the library Computer Assisted Proof in Dynamic groups (CAPD) [50] which algorithm is based on an enhanced Löhner algorithm. We will compare ourselves to it, but also to the Löhner algorithm implemented in Codac. There exists numerous of other solvers such as VNODE ([94]), DynIbex ([1]) or Valencia-IVP ([101])

3.4 Löhner's algorithm

As mentioned in the previous section, we will use CAPD based on the Löhner algorithm to compute guaranteed integration and also to compare it to the method presented in Chapter 5. In this section we will present briefly the method developed by Löhner in [82].

3.4.1 Rigorous integration

We aim at solving the IVP defined by Equation (3.6) where $\mathbf{f} \in \mathcal{C}^p(\mathbb{R}^n)$. We denote by $\phi(t, \mathbf{x})$ the associated flow function.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases} \tag{3.6}$$

Löhner's algorithm was designed to perform guaranteed integration of Equation (3.6) in the same manner as the simple Euler scheme but in a rigorous way. Hence the result will be a gathering of successively computed intervals $([x_k])_{k\in\mathbb{N}}$, where k denotes the integration time t_k such that $\forall \mathbf{x}_0 \in [\mathbf{x}_0], \phi(t_k, \mathbf{x}_0) \in [\mathbf{x}_k]$. An illustration of this is given in Figure 3.1

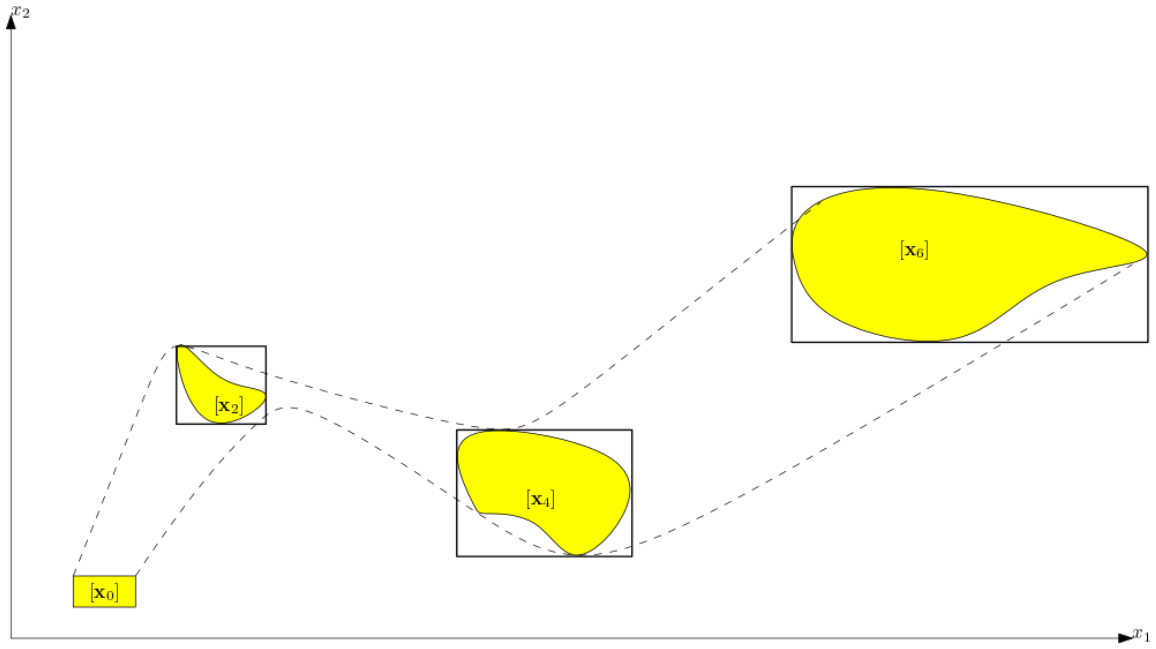


Figure 3.1: Rigorous integration of an IVP.

3.4.2 Taylor-Lagrange expansion

As in most integration schemes, each interval $[x_k]$ is computed using the value of $[x_{k-1}]$. To compute each step from the previous one, Löhner based his algorithm on the Taylor-Lagrange expansion. Let $\mathbf{x}(t)$ be the solution of the IVP such that $\mathbf{x}(0) = \mathbf{x}_0 \in [\mathbf{x}_0]$ where

$[\mathbf{x}_0] = \mathbf{x}_0 + [\mathbf{z}_0]$. Let $\mathbf{x}_k$ denote $\mathbf{x}(t_k)$ and h the integration step $t_{k+1} - t_k$. As $\mathbf{f} \in \mathcal{C}^p(\mathbb{R}^n)$ we can apply the Taylor-Lagrange expansion to the p^{th} order to the dynamical system related to $\phi(t, \mathbf{x}_0)$. Doing so at time t_k, we obtain:

$$\phi_{\mathbf{x}_0}(t_k + h) = \phi_{\mathbf{x}_0}(t_k) + \sum_{i=1}^{p-1} \frac{h^i}{i!} \frac{\partial^i \phi_{\mathbf{x_0}}}{\partial t^i}(t_k) + \mathbf{z}_{k+1}, \tag{3.7}$$

where $\mathbf{z}_{k+1}$ denotes the Lagrange remainder. It is also called the *discretisation error* as it corresponds to the difference between the true solution and the Taylor series approximation. To simplify the reading, Equation (3.7) can be rewritten:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sum_{i=1}^{p-1} \frac{h^i}{i!} \mathbf{x}_k^{(i)} + \mathbf{z}_{k+1}. \tag{3.8}$$

Now, the expression given in Equation (3.8) yields an exact solution. Our goal is to find an enclosure $[\mathbf{x}_{k+1}]$ such that $\mathbf{x}_{k+1} \in [\mathbf{x}_{k+1}]$. An enclosure of the sum must be computed but will not be detailed here. The reader may refer to [14, Chapter 2.4.5] for further details.

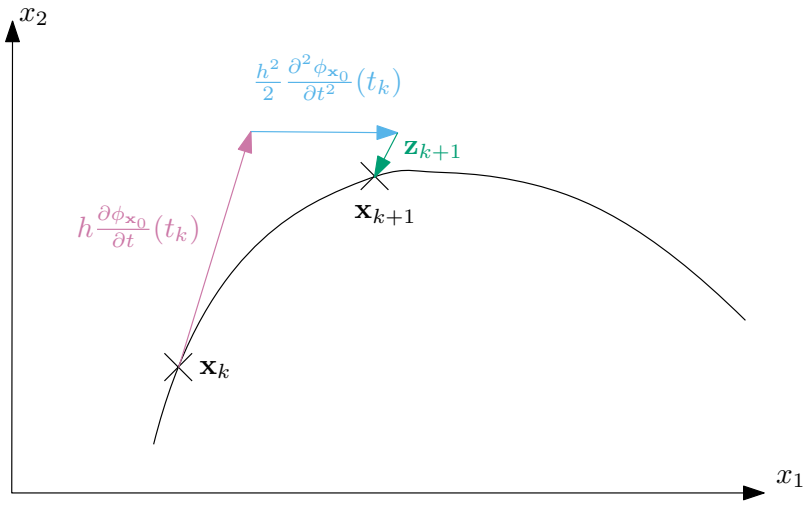


Figure 3.2: Taylor-Lagrange expansion to the third order.

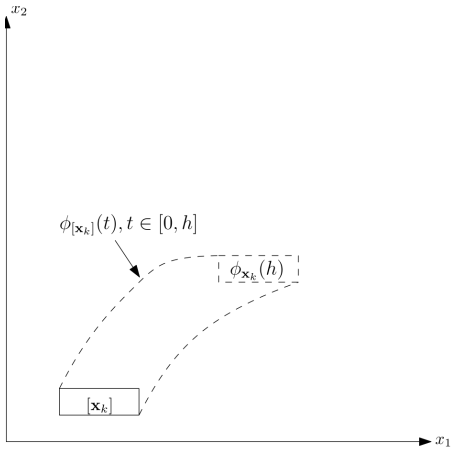
3.4.3 Löhner's algorithm steps

The algorithm is based on two main steps [25]. A first one that can be seen as a prediction step. It consists in finding a coarse enclosure $[\tilde{\mathbf{x}}_k]$ which is an enclosure of $\mathbf{x}(t)$ over the time $[t_k, t_{k+1}]$. This coarse enclosure is called *global enclosure* and satisfies the following properties [124]:

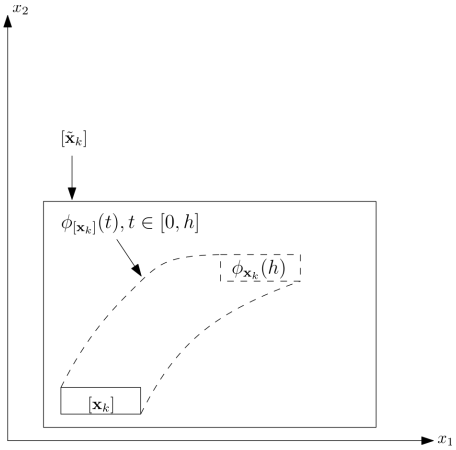
- $\mathbf{x}(t; [\mathbf{x}_k])$ (which means $\mathbf{x}(\cdot)$ passing through $[\mathbf{x}_k]$) is guaranteed to exist for all $t \in [t_k, t_{k+1}]$,
- $\mathbf{x}(t; [\mathbf{x}_k]) \subseteq [\tilde{\mathbf{x}}_k]$ for all $t \in [t_k, t_{k+1}]$,

- the step-size $h = t_{k+1} - t_k > 0$ is as large as possible while keeping a correct accuracy (defined by the user) and existence proof of the IVP solution.

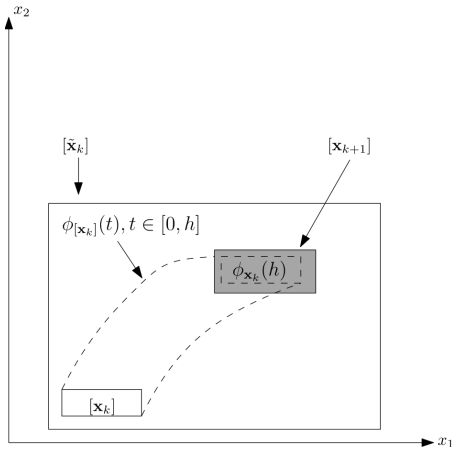
Then a second step is performed which comes down to using the latter enclosure to compute a local enclosure $[\mathbf{x}_{k+1}]$ of $\mathbf{x}_{k+1}$, such that $\mathbf{x}(t_{k+1}; [\mathbf{x}_k]) \subseteq [\mathbf{x}_{k+1}]$. This second step could be considered as the correction step as it contracts the enclosure $[\tilde{\mathbf{x}}_k]$. Indeed by computing a local enclosure of $[\mathbf{x}_{k+1}]$ at time t_{k+1} we have a constraint on $[\tilde{\mathbf{x}}_k]$ as by definition $[\mathbf{x}_{k+1}] \in [\tilde{\mathbf{x}}_k]$, which allows us to contract it. These two steps can be repeated again and again until reaching a fixed point to obtain a better enclosure $[\tilde{\mathbf{x}}_k]$. Figure 3.3 illustrates these steps. We will only detail the first step in the next section to underline one main drawback of such a method. The second part is straightforward using Equation (3.8), again the reader may refer to [14] for a detailed reasoning.



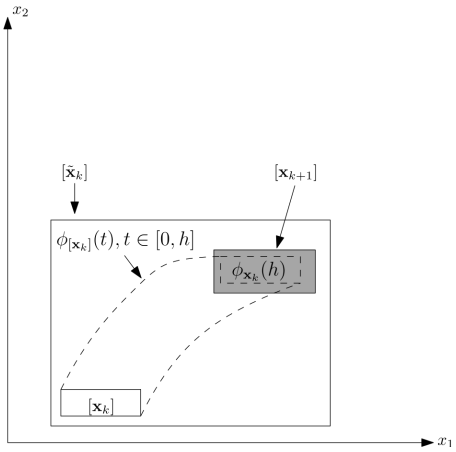
(a) Initial setup



(b) Step 1, a global enclosure $[\tilde{\mathbf{x}}_k]$ is computed (see Section 3.4.4).



(c) Step 2, computation of a local enclosure $[\mathbf{x}_{k+1}]$.



(d) Step 3, the global enclosure $[\tilde{\mathbf{x}}_k]$ is contracted.

Figure 3.3: The different steps of Löhner's algorithm.

3.4.4 Global enclosure

Let us suppose that we are able to compute the solution up to time t_k, then we have everything needed to compute $[\mathbf{x}_{k+1}]$ as it depends only on $[\mathbf{x}_k]$ except $\mathbf{z}_{k+1}$. This discretisation error due to the Taylor-Lagrange expansion, depends on the true solution $\mathbf{x}(\cdot)$ evaluated at an unknown time $\tau \in [t_k, t_{k+1}]$. As it is not possible to evaluate this quantity directly, Löhner proposes to compute a global enclosure of $\mathbf{x}(t)$, $\tilde{\mathbf{x}}_k$ over the time interval $[t_k, t_{k+1}]$, i.e

$$[\tilde{\mathbf{x}}_k] = [\{\mathbf{x}(t) | t \in [t_k, t_{k+1}], \mathbf{x}(t_k) \in [\mathbf{x}_k]\}] \tag{3.9}$$

This notion of global enclosure has been introduced by Moore in [90] before being further developed in [91, Chapter 10]. This method called First Order Enclosure (FOE), is based on the application of the Banach fixed point theorem [6] and the Picard-Lindelöf operator that we recall below.

Theorem 3.1 (Banach fixed-point theorem)**.** *Let (K, d) a complete metric space and let $g : K \to K$ a contraction that is for all $x, y \in K$ there exists $c \in]0, 1[$ such that $d(g(x), g(y)) \leq d(x, y)$, then g has unique fixed-point in K*

Definition 3.1 (Picard-Lindelöf operator)**.** Consider the system defined by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$ and let us denote $\mathbf{x}_k = \mathbf{x}(t_k)$ the solution of Equation (3.6) at time t_k, then its Picard-Lindelöf operator is defined as:

$$\mathbf{p}_{\mathbf{f}\mathbf{x}_k}(t) = \mathbf{x}_k + \int_{t_k}^{t} \mathbf{f}(\mathbf{x}(u))du \tag{3.10}$$

This operator being associated with the integral form of Equation (3.6), if it is a contraction operator then its solution is unique and is solution of Equation (3.6). It is possible to define the interval counterpart of the Picard-Lindelöf operator using a simple first order integration scheme (hence the name FOE). Let $\hat{\mathbf{x}}_{k,t}$ be a candidate enclosure of $\mathbf{x}(\tau), \forall \tau \in [t_k, t]$, then we have:

$$\int_{t_k}^{t} \mathbf{f}(\mathbf{x}(u))du \in (t - t_k)\mathbf{f}([\tilde{\mathbf{x}}_{k,t}]). \tag{3.11}$$

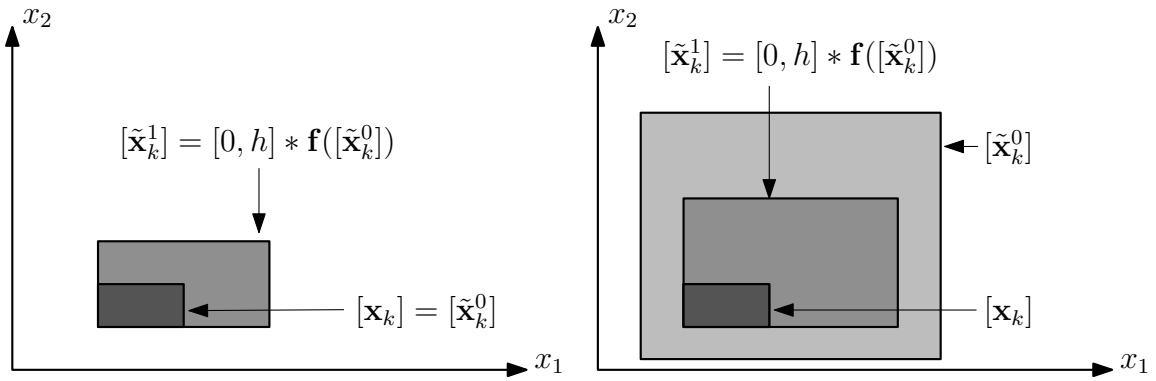
Now let $[\tilde{\mathbf{x}}_k^0]$ be a first candidate enclosure of $[\tilde{\mathbf{x}}_k]$ (Equation (3.9)). Applying Equation (3.11) on the Picard-Lindelöf operator at $t = t_{k+1}$ we obtain

$$\begin{aligned}
\mathbf{p}_{\mathbf{f}\mathbf{x}_k}(t_{k+1}) &= \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(\mathbf{x}(u))du && (3.12)\\
&\in [\mathbf{x}_k] + \int_{t_k}^{t_{k+1}} \mathbf{f}([\hat{\mathbf{x}}_{k,t_{k+1}}])du && (3.13)\\
&\overset{(3.9)}{\in} [\mathbf{x}_k] + \int_{t_k}^{t_{k+1}} \mathbf{f}([\tilde{\mathbf{x}}_k^0])du && (3.14)\\
&\subseteq [\mathbf{x}_k] + [0, h]\mathbf{f}([\tilde{\mathbf{x}}_k^0]) = [\tilde{\mathbf{x}}_k^1], && (3.15)
\end{aligned}$$

where $h = t_{k+1} - t_k$ is the step-size.

Now, from the Banach fixed-point theorem, if $[\tilde{\mathbf{x}}_k^1] \subseteq [\tilde{\mathbf{x}}_k^0]$ then it exists a trajectory $\phi_{\mathbf{x}_k}(\cdot)$ such that $\phi_{\mathbf{x}_k}(t) \in [\tilde{\mathbf{x}}_k^1]$ for all $t \in [t_k, t_{k+1}]$ and all $\mathbf{x}_k \in [\mathbf{x}_k]$. In addition, the theorem

ensures that this solution is unique. Now the question is: "How do we choose $[\tilde{\mathbf{x}}_k^0]$?". To answer it, Löhner proposes a simple algorithm presented in Algorithm 1. It consists in defining $[\tilde{\mathbf{x}}_k^0] = [\mathbf{x}_k]$ during the first iteration and computing $[\tilde{\mathbf{x}}_k^1]$. If $[\tilde{\mathbf{x}}_k^1] \not\subseteq [\tilde{\mathbf{x}}_k^0]$ then $[\tilde{\mathbf{x}}_k^0]$ is inflated by a factor α and the process is repeated until $[\tilde{\mathbf{x}}_k^1] \subseteq [\tilde{\mathbf{x}}_k^0]$. Now, this method might not work depending on the value of h. If it is too large, a global enclosure might not be found. Thus the algorithm proposed also reduces the step-size h by a factor $\nu \in]0, 1[$ if a global enclosure could not be found after a number N of iterations.



(a) $[\tilde{\mathbf{x}}_k^1] \not\subset [\tilde{\mathbf{x}}_k^0]$, we need at least another iteration

(b) $[\tilde{\mathbf{x}}_k^1] \subset [\tilde{\mathbf{x}}_k^0]$, the algorithm terminates

Figure 3.4: Global enclosure algorithm

Algorithm 1 GlobalEnclosure(in: $[\mathbf{x}_k]$, h, N, ν, out: $[\tilde{\mathbf{x}}_k]$, h)

$[\tilde{\mathbf{x}}_k^0] \leftarrow [\mathbf{x}_k]$
$[\tilde{\mathbf{x}}_k^1] \leftarrow [\mathbf{x}_k] + [0, h]\mathbf{f}([\tilde{\mathbf{x}}_k^0])$
$n \leftarrow 0$
while $[\tilde{\mathbf{x}}_k^1] \not\subseteq [\tilde{\mathbf{x}}_k^0]$ **do**
  $n \leftarrow n + 1$
  if $n \geq N$ **then**
    $n \leftarrow N$
    $h \leftarrow \nu h$

    $[\tilde{\mathbf{x}}_k^0] \leftarrow [\mathbf{x}_k]$
    $[\tilde{\mathbf{x}}_k^1] \leftarrow [\mathbf{x}_k] + [0, h]\mathbf{f}([\tilde{\mathbf{x}}_k^0])$
  end if
  $[\tilde{\mathbf{x}}_k^0] \leftarrow (1 + \alpha)[\tilde{\mathbf{x}}_k^1] - \alpha[\tilde{\mathbf{x}}_k^1]$
  $[\tilde{\mathbf{x}}_k^1] \leftarrow [\mathbf{x}_k] + [0, h]\mathbf{f}([\tilde{\mathbf{x}}_k^0])$
end while
return $[\tilde{\mathbf{x}}_k^0]$, h

3.4.5 Löhner's algorithm limits

From what was presented in Section 3.4, we can underline some limits to the algorithm Lohner proposed.

1. The first one being the conventional step-by-step structure. Indeed, if one want to compute the trajectory $\phi_{[\mathbf{x}_0]}(t)$ up to time t_f, one might need to compute numerous steps from t_0 to t_f. If one has to perform guaranteed integration for multiple $[\mathbf{x}_0]$ this can take a tremendous amount of time and potentially require a huge amount of computational power.

2. Then as mentioned at the beginning of this section, the use of axis-aligned induces some wrapping effect. This phenomenon can quickly become an issue after just a few steps of computations as it is shown in Figure 3.5 which presents the example of the rotation of a box.

3. The third limit is the computation of the global enclosure. This step is the source of two problems. Firstly, if a global enclosure cannot be found then the whole guaranteed integration fails. Then, the algorithm may repeat the same operations several times in order to compute the global enclosure, increasing greatly the need for computational power and time.

4. The last one is the use of the centered form of $\mathbf{f}$ presented in Section 2.3.3.2. If this form has the advantage of getting close to the minimal inclusion function of $\mathbf{f}$, its efficiency is heavily dependent on the size of the interval/box computed. Hence, if the interval/box is too large, one can experience a bloating effect (see Figure 3.6) or the algorithm might not be able to compute the global enclosure thus failing completely in performing the guaranteed integration. This greatly limits the size of the box taken as initial condition, which is, in a robotics context, a major drawback as our uncertainties may be quite important.

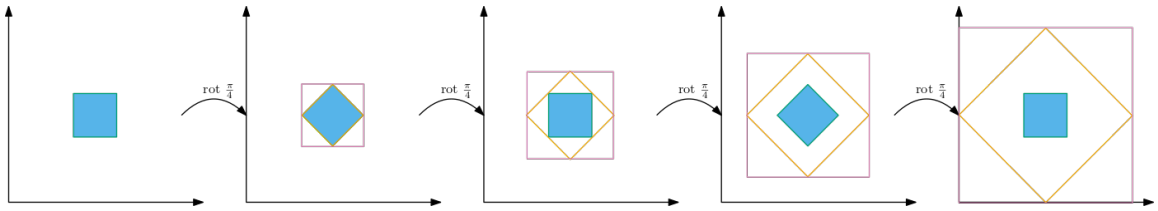


Figure 3.5: Wrapping induced when integrating the rotation of a box. In blue is represented the true result, in pink the guaranteed enclosure computed and in orange the enclosure of the previous step after application of the rotation

3.4.6 Enhance Löhner's algorithm

In order to enhance the efficiency of the algorithm presented by Löhner, some strategies have been developed, they tackle two main issues.

1. The first group focuses on keeping the step-size as large as possible, thus reducing the number of steps needed to perform the integration. To do so, they improve the global enclosure research method as it is the one that affects the step-size, reducing it. Most of them are based on using a *high-order enclosure* (as opposed to FOE) as presented in [25, 95]

2. The second one addresses the wrapping effect problem. Indeed as shown in Figure 3.5, the pessimism induced can quickly become an issue when performing the integration. If the initial condition at time t_0 we have and the final result we want at time t_f may be axis-aligned boxes, the representation of the sets during the computation steps can be different. Several representations have been developed, namely parallelepipeds [69, 89], cuboids [82], doubletons [82] or tripletons [50, 66]. When choosing a representation, one should consider the trade-off between efficiency and computational power required. An exhaustive review of this different representations is available in [14].

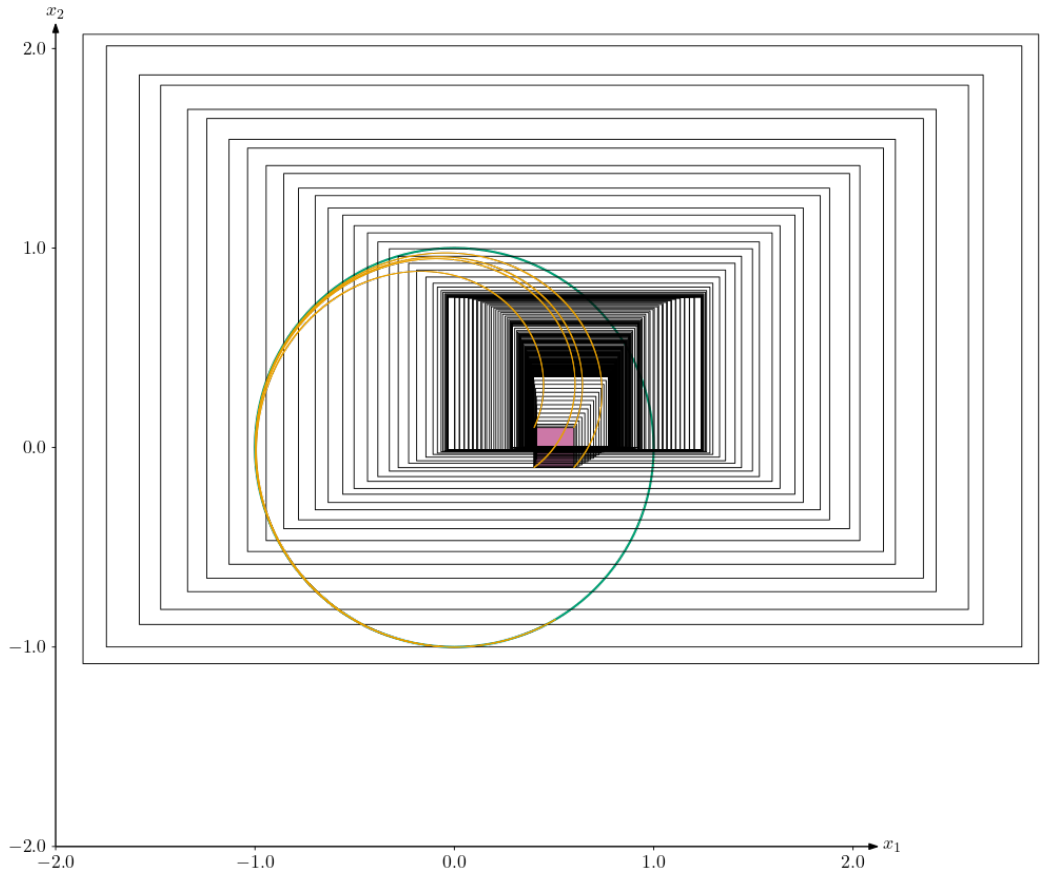


Figure 3.6: Bloating effect: We decided to perform a guaranteed integration with CAPD and the system presented in Equation (2.5), using as initial condition the box $[\mathbf{x}] = [0.4, 0.6] \times [-0.1, 0.1]$ (painted pink) over the time interval $[0, 5]$. This results in very pessimistic enclosure (painted black) after a few steps (the algorithm stops at time t = 0.7). However when using as initial condition the corners of the box, the algorithm performs well, with sharp enclosures of the trajectories represented in gold. They converge towards the unit circle painted green. Thus the size of the initial box is a critical parameter when using this kind of algorithm.

3.5 Perform guaranteed integration with Codac

Codac provides two tools to perform guaranteed integration. The first one is based on a simplified Löhner algorithm turned into a contractor on a tube. The full details of this contractor implementation are given in [14]. The other method is one small contribution of this thesis. In order to benefit from the years of development and improvements made by the CAPD team, a binding has been made in Codac to perform guaranteed integration with CAPD and have a Codac Tube (or TubeVector) object as output. This enables us to use tools presented in Chapter 2 while benefiting from the speed and accuracy of CAPD computations. Let us present them in an example

Example 3.2 (Guaranteed integration with Codac)**.** Consider the system defined by

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} 1 \\ -x_2 \end{pmatrix}. \tag{3.16}$$

We have as initial condition the box $[\mathbf{x}_0] = [\mathbf{x}](0) = \begin{pmatrix} [0, 0.1] \\ [2, 2.1] \end{pmatrix}$. We would like to compute $[\mathbf{x}](5)$. We first did it using the integrated Löhner contractor first. The piece of code to do it is provided in Listing 3.1. We then performed the same operation using the binding with CAPD (Listing 3.2). The results of both integration are displayed in Figure 3.7. One can notice that the tube enclosing the possible trajectories obtained with CAPD is thinner than the one obtained with the Löhner contractor as expected. Not only the result obtained with CAPD is more accurate, it is also faster to compute. It only takes 3 ms to perform the integration with CAPD when the Löhner contractor takes 319 ms *i.e* it is a hundred times slower. Of course this result gets worse with the complexity of the system to integrate and the length of the time domain we are integrating on. The computer used processor used is an Intel I7 8650U @ 1.90GHz.

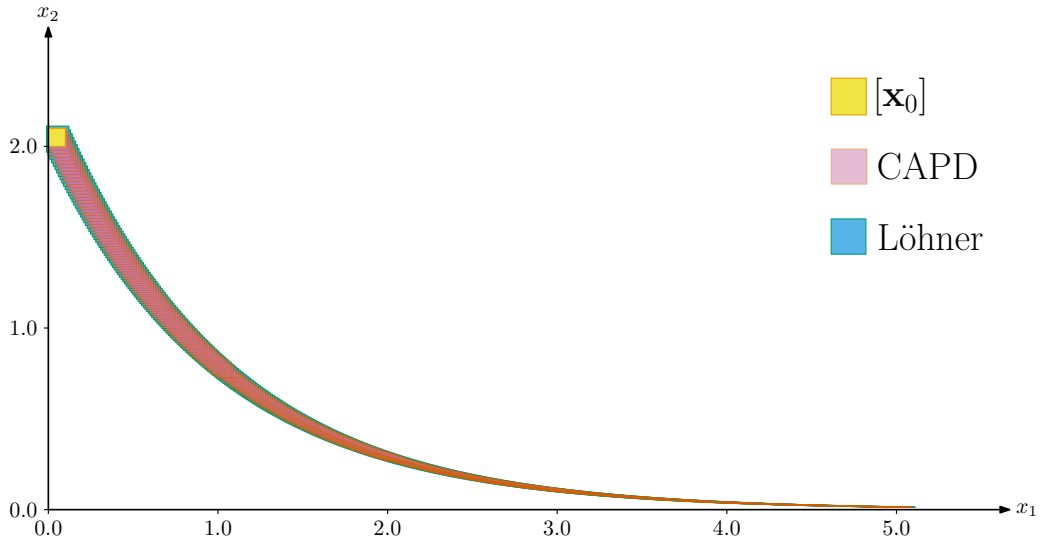


Figure 3.7: Result of guaranteed integration using CAPD or the Löhner contractor

Listing 3.1 Perform guaranteed integration using the Löhner algorithm integrated in Codac V1

```
Interval time_domain(0,5); // Integration time
double h = 0.01; // Time step for the integration scheme
IntervalVector x_0({{0,0.1},{2,2.1}}); // Initial condition
Function f("x1","x2","(1; -x2)"); // Evolution function to integrate
/*
 * Creation of a default tube: tube(time domain, time step, nb of dimensions)
 * For each dimension [a][i](.)(t)= [-inf,inf]
 */
TubeVector a_Lohner(time_domain,h,2);
// Creation of the Lohner contractor for the system defined by f
CtcLohner c_Lohner(f);
// Setting the initial condition: a.set(box, time)
a_Lohner.set(x_0,0.);
// Performing the integration
c_Lohner.contract(a_Lohner);
```

Listing 3.2 Perform guaranteed integration using the binding with CAPD (Codac V1)

```
Interval time_domain(0,5); // Integration time
double h = 0.01; // Time step for the integration scheme
IntervalVector x_0({{0,0.1},{2,2.1}}); // Initial condition
Function f("x1","x2","(1; -x2)"); // Evolution function to integrate

// Generating the output tube of the integration using CAPD
TubeVector a_capd = CAPD_integrateODE(time_domain,f,x_0,h);
```

CHAPTER 4

LIE GROUPS AND DIFFERENTIAL EQUATIONS

Contents

4.1 Lie Group Theory

We have seen in the previous section that the existing tools dedicated to guaranteed integration struggle when the dynamical system we are considering is highly non linear and the initial condition is rather large. To tackle this problem, in the next chapter, we will use Lie symmetries. This section will introduce the basics of Lie group theory and serve as a crash course for the rest of this thesis.

4.1.1 Lie group: a smooth manifold ...

The first element needed to have a lie group is a *smooth manifold* (also called differentiable manifold). It is a topological space that locally resembles a linear space. To ease the reader understanding, an example is provided in Figure 4.1. It is a curved, smooth space, with no spikes or discontinuities of any kind, embedded in a space of higher dimensions. In Figure 4.1, it is the unit circle embedded in $\mathbb{C}$. The unit circle locally resembles $\mathbb{R}$. The differentiability of the space ensures that there exists a unique tangent space at each point of the manifold. This tangent space is linear and it is possible to perform calculus on it.

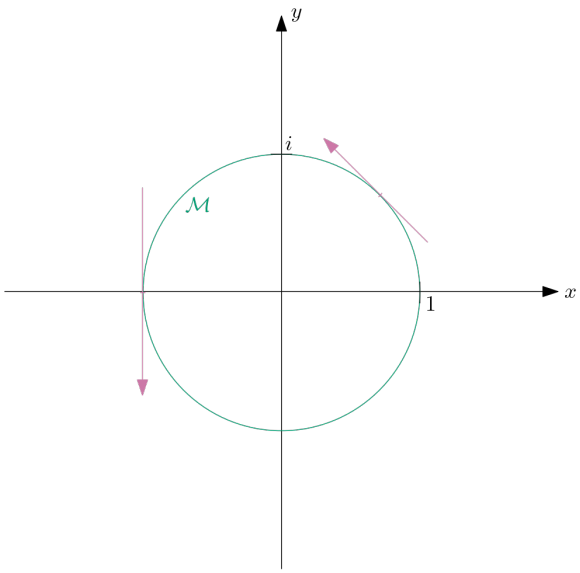


Figure 4.1: The unit circle is a smooth manifold of $\mathbb{C}$

4.1.2 ... following group axioms

Lie groups are strongly related to the mathematical notion of groups, let us recall the latter through definitions and examples.

Definition 4.1 (Group)**.** A group denoted $(G, \circ)$ is a composed of a set G, associated with a binary operation $\circ$ defined on G which satisfies the following rules:

- Closure: $\forall g_1, g_2 \in G, g_1 \circ g_2 \in G$.

- Associativity: $\forall g_1, g_2, g_3 \in G, g_1 \circ (g_2 \circ g_3) = (g_1 \circ g_2) \circ g_3$.
- There exists an identity element $e \in G$ such that $\forall g \in G, g \circ e = e \circ \ g = g$.
- $\forall g \in G$ there exists $g^{-1} \in G$ such that $g \circ g^{-1} = g^{-1} \circ g = e$.

To clarify this notion we will use the example of the group $(S_3, \circ)$ composed of the set of all permutations on a 3-element set and the composition rule. Let us recall the definition of a permutation first.

Definition 4.2 (Permutation)**.** A permutation σ on a set E is a bijection $E \to E$. It can be seen as a mapping of the elements of E on themselves.

Example 4.1 (Permutation)**.** Let E be a set composed of 3 elements $E = \{a, b, c\}$. Let σ be a permutation on E such that

$$\sigma = \begin{pmatrix} a & b & c \\ a & c & b \end{pmatrix}.$$

Hence $\sigma(a) = a$, $\sigma(b) = c$ and $\sigma(c) = b$.
We can also take all the elements of E in a particular order for example (a,b,c) and apply the permutation σ. In that case, $\sigma(a, b, c) = (a, c, b)$. For the rest of this chapter we will write permutations of 3 elements in the following manner $\sigma : abc \to acb$.

Example 4.2 (Group)**.** We will show that $(S_3, \circ)$ satisfies all the characteristics of the group.
Firstly:

$$S_3 = \begin{Bmatrix} \sigma_1 : abc \to abc, & \sigma_2 : abc \to acb, & \sigma_3 : abc \to bac, \\ \sigma_4 : abc \to cba, & \sigma_5 : abc \to bca, & \sigma_6 : abc \to cab \end{Bmatrix}$$

- Closure: It is easy to demonstrate that $\forall \sigma_1, \sigma_2 \in S_3, \sigma_1 \circ \sigma_2 \in S_3$. Any composition of two sortings of three elements will give a sorting of three elements. For example $\sigma_2 \circ \sigma_3 = \sigma_5$.
- Associativity: $\forall \sigma_i, \sigma_j, \sigma_k \in S_3, (\sigma_i \circ \sigma_j) \circ \sigma_k = \sigma_i \circ \sigma_j \circ \sigma_k = \sigma_i \circ (\sigma_j \circ \sigma_k)$.
- $\forall \sigma_i \in S_3, \sigma_1 \circ \sigma_i = \sigma_i \circ \sigma_1 = \sigma_i$ Thus there exists a neutral element here denoted σ_1.
- $\forall i \in \{1, 2, 3, 4\}, \sigma_i \circ \sigma_i = \sigma_1$ and $\sigma_5 \circ \sigma_6 = \sigma_6 \circ \sigma_5 = \sigma_1$. Hence each element of S_3 has an inverse element in S_3 by $\circ$.

Therefore, the set S_3 of all permutations in a set of 3 elements, associated with the composition rule $\circ$ forms a group $(S_3, \circ)$.

4.1.3 Definitions

With the notions of manifold and group introduced, it is now possible to define Lie Groups.

Definition 4.3 (Lie Group)**.** A Lie group is a group G which also carries the structure of a smooth manifold such that the group operations are differentiable maps on this manifold.

Definition 4.3 means that the set $\mathcal{M}$ of the Lie group $(\mathcal{M}, \diamond)$ is a smooth manifold and that the group axioms are valid on this set using the operator $\diamond$. The $\diamond$ operator is not necessarily the composition as we will see in the examples. Thus it exists an identity element $\mathcal{E} \in \mathcal{M}$ and for all $\mathcal{X} \in \mathcal{M}$, there exists $\mathcal{X}^{-1} \in \mathcal{M}$ such that $\mathcal{X} \diamond \mathcal{X}^{-1} = \mathcal{X}^{-1} \diamond \mathcal{X} = \mathcal{E}$. The closure and associativity properties are also satisfied. To ease the notation, a Lie group $(\mathcal{M}, \diamond)$ is generally shortened to the name of its manifold $\mathcal{M}$.

As the Lie group encapsulates the notion of smooth manifold, for each element $\mathcal{X}$ of a Lie group $\mathcal{M}$, there exists a tangent linear vector space at this element denoted $T_{\mathcal{X}}\mathcal{M}$. Among all these tangent vector spaces, the one at the identity element $\mathcal{E}$, $T_{\mathcal{E}}\mathcal{M}$, is called the *Lie algebra* of the Lie group $\mathcal{M}$ and is denoted $\mathfrak{m}$. The main interest of such tangent spaces is that there linear structure allows easier computations.

To go from the Lie group to its associated algebra and inversely, two operators exist. On the one hand, the *exponential map*, links all elements of the Lie algebra to the elements of the group, $\exp : \mathfrak{m} \to \mathcal{M}$. On the other hand the *log map* transforms elements of the Lie group into elements of the Lie algebra $\log : \mathcal{M} \to \mathfrak{m}$.

Example 4.3 (Lie group)**.** A simple example often used to illustrate Lie groups is the unit complex numbers group equipped with complex multiplication S^1 [110]. Each element of this group can be written under the form $\mathbf{z} = cos(\theta) + isin(\theta)$.

- Group properties:
  - (closure) Let u, v be two unit complex numbers, $u \cdot v \in S^1$,
  - (identity) 1 is the identity element,
  - (inverse) $u \in S^1$ and its complex conjugate u^*, $u \cdot u^* = u^* \cdot u = 1$,
  - (associativity) $u, v, w \in S^1$, $(u \cdot v) \cdot w = u \cdot (v \cdot w)$.
- Manifold properties: we have seen in Section 4.1.1 that the unit circle was a smooth manifold in the complex plane.
- Group action (see Section 4.1.4): Consider $\mathbf{z} \in S^1$. Each vector $\mathbf{x}$ of the form $\mathbf{x} = x + iy$ is rotated in the plane by an angle θ through complex multiplication $\mathbf{x}' = \mathbf{zx}$.

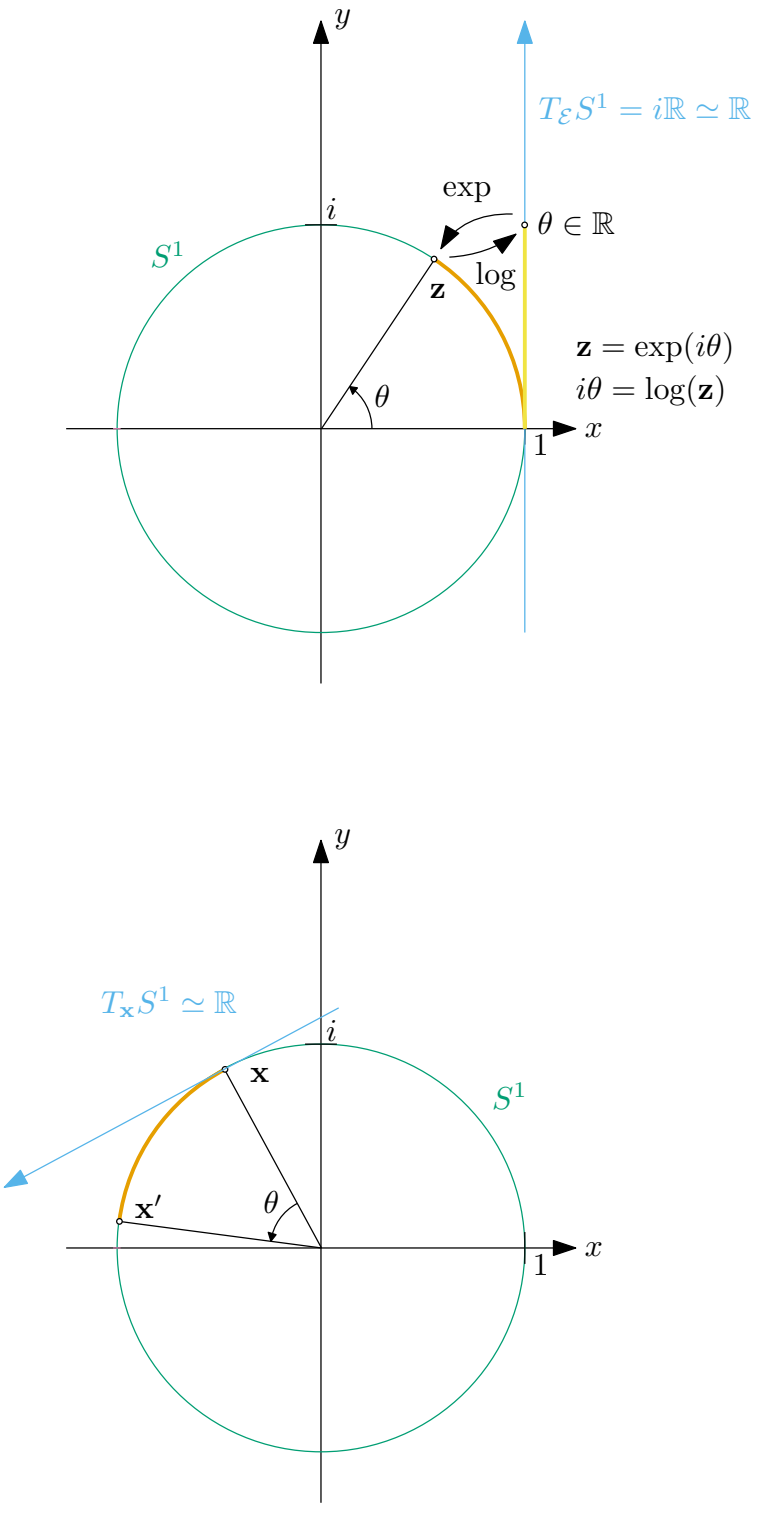


Figure 4.2: The Lie group S^1

4.1.4 Group action

Lie groups are mainly used for their ability to transform elements of other sets, for instance rotations, translations, scalings or combinations of them. These operations are performed on a daily basis in the robotic field as they describe a robot trajectory in 2D and 3D. However, the use of this theory in the robotics field is quite recent [7, 36, 110, 112].

Definition 4.4. (Group action) Consider a group $(\mathcal{M}, \diamond)$ and a set $\mathcal{F}$, the left group action of $\mathcal{X} \in \mathcal{M}$ on $f \in \mathcal{F}$ is the operation denoted $\mathcal{X} \bullet f$.

$$\bullet : \begin{array}{ccc} \mathcal{M} \times \mathcal{F} & \rightarrow & \mathcal{F} \\ (\mathcal{X}, f) & \mapsto & \mathcal{X} \bullet f \end{array} \tag{4.1}$$

The operator $\bullet$ also satisfies the axioms:

1. (Identity) Consider $\mathcal{E}$ the identity element of $(\mathcal{M}, \diamond)$ then $\forall f \in \mathcal{F}, \mathcal{E} \bullet f = f$
2. (Compatibility) Consider $\mathcal{X}, \mathcal{Y} \in \mathcal{M}$ then $\forall f \in \mathcal{F}, (\mathcal{X} \diamond \mathcal{Y}) \bullet f = \mathcal{X} \bullet (\mathcal{Y} \bullet f)$

Remark 4.1. The *right group action* also exists, denoted $f \bullet \mathcal{X}$ but will not be of use here.

Example 4.4. Consider a set $\mathbb{A} = a, b, c$ and $\mathbb{F}$ the set of all applications from $\mathbb{A}$ to $\mathbb{A}$. Let us pick $f \in \mathbb{F}$ such that $f(a) = a,\ f(b) = a,\ f(c) = b$. Then $\forall \sigma_i \in S_3$ (see Example 4.2) define the action of σ_i on f as:

$$\sigma_i \bullet f = f \circ \sigma_i. \tag{4.2}$$

It is trivial to check the identity property such that that $\sigma_1 \bullet f = f$. Then let us verify that it satisfies the compatibility axiom.

$$\begin{aligned} (\sigma_i \circ \sigma_j) \bullet f &= f \circ (\sigma_i \circ \sigma_j) \\ &= (f \circ \sigma_i) \circ \sigma_j \\ &= (\sigma_i \bullet f) \circ \sigma_j \\ &= \sigma_j \bullet (\sigma_i \bullet f). \end{aligned}$$

Thus the compatibility property holds. For instance, let us define the action of σ_5 on f such that $g = \sigma_5 \bullet f = f \circ \sigma_5$. Then $g(a) = \sigma_5 \bullet f(a) = f \circ \sigma_5(a) = f(c) = b$. Hence the action of σ_5 converts the transformation f into another one. Figure 4.3 depicts this conversion. The transformation f is represented as a dynamical graph as we will consider dynamical systems later in this thesis.

Example 4.5. In a robotic context, the main Lie groups of interest, associated with transformations mentioned in Definition 4.4 above, are the groups of rotation matrices denoted $SO(n)$ and the group of rigid motions $SE(n)$ with n being the dimension of the problem (usually 2 or 3). In this cases, there group actions on a state vector $\mathbf{x} \in \mathbb{R}^n$ can be defined as follows:

- Consider $\mathbf{R} \in SO(n), \mathbf{x} \in \mathbb{R}^n, n \in \{2, 3\}$, then $\mathbf{R} \bullet \mathbf{x} = \mathbf{R}\mathbf{x}$.
- Consider $\mathbf{H} \in SE(n), \mathbf{x} \in \mathbb{R}^n, n \in \{2, 3\}$, then $\mathbf{H} \bullet \mathbf{x} = \mathbf{R}\mathbf{x} + \mathbf{t}$ ($\mathbf{t}$ represents a translation and not time.).

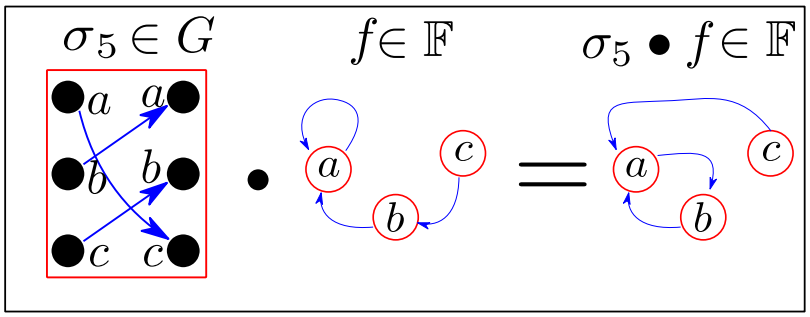


Figure 4.3: A permutation σ_5 of S_3 acting on an object $f \in \mathbb{F}$

With group actions presented, we introduce two other notions linked to them used in the Section 4.3.

Definition 4.5 (Orbit)**.** Consider a group G acting on a set $\mathbb{F}$. The *orbit* of an element $f \in \mathbb{F}$ is $G(f) = \{g \bullet f | g \in G\}$, the minimal subset of $\mathbb{F}$ that contains all the transformations of f by $g \in G$.

Definition 4.6 (Stabiliser)**.** Consider an element $f \in \mathbb{F}$. We define the stabiliser subgroup $\text{Sym}_G(f)$ of G with respect to f the set of all elements of G for which f remains unchanged, *i.e.*:

$$\text{Sym}_G(f) = \{\sigma \in G | \sigma \bullet f = f\}. \tag{4.3}$$

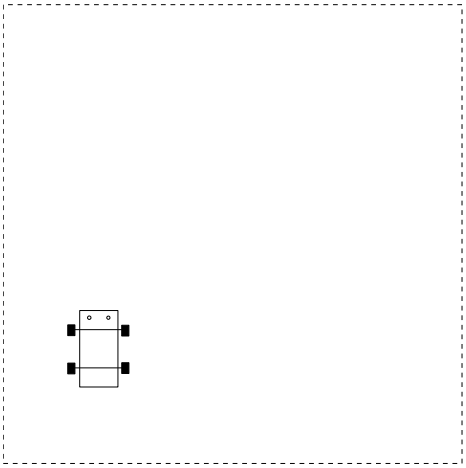
Any element of $\text{Sym}_G(f)$ is called a *stabiliser* of f.

Example 4.6. (Stabiliser) Let us consider again the elements introduced in Example 4.4. As $\sigma_1(a, b, c) = (a, b, c)$, $\sigma_1 \bullet f = f$, hence $\sigma_1 \in \text{Sym}_G(f)$. From Example 4.4, we can conclude that $\sigma_5 \notin \text{Sym}_G(f)$. Applying the group action on f using $\sigma_2, \sigma_3, \sigma_4$ and σ_6, it is easy to check that apart from σ_1 only $\sigma_3 \bullet f = f$. Therefore:

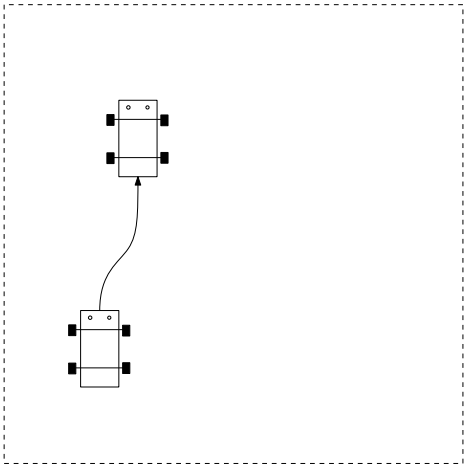
$$\text{Sym}_G(f) = \{\sigma_1, \sigma_3\}. \tag{4.4}$$

4.2 General idea of the new integration method

The idea behind the work developed in this chapter and the next one is the following. Let us suppose we have an object, here a car, as shown in Figure 4.4a. This object moves, during one second, following the trajectory depicted by the black arrow in Figure 4.4b. Let us move this situation in the mathematical domain saying that the first car is at some position $\mathbf{x}_0^1$ (the exponent is an index here). The evolution function $\mathbf{f}$ associated with our object is available and we perform a guaranteed integration. The result obtained, denoted $\phi_1(\mathbf{x}_0^1)$, is the one expected and the different 4 steps of the integration schemes are depicted by coloured arrows in Figure 4.4c. Now we would like to know the final position of another object placed at $\mathbf{x}_0^2$ depicted in green on Figure 4.4d.

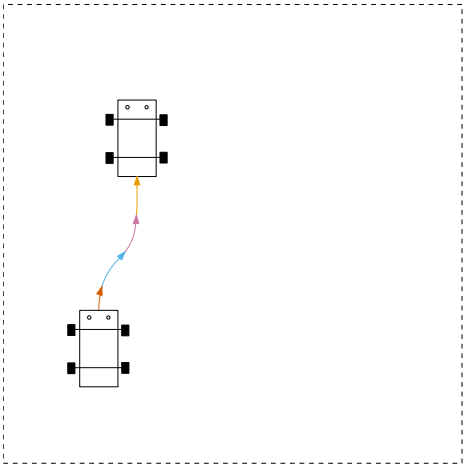


(a) initial position

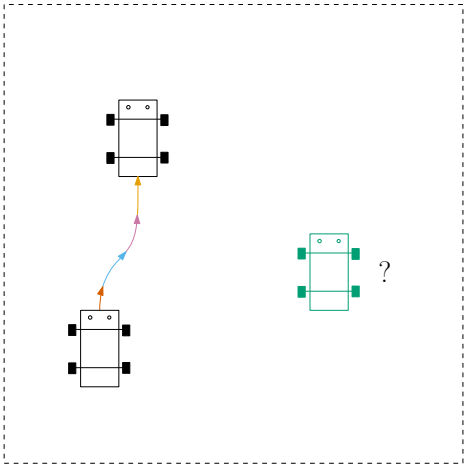


(b) position after 1 second

Figure 4.4: Illustration of the integration principle



(c) Guaranteed integration steps

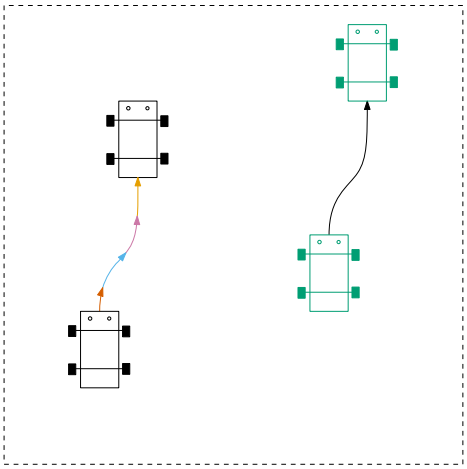


(d) Second initial position

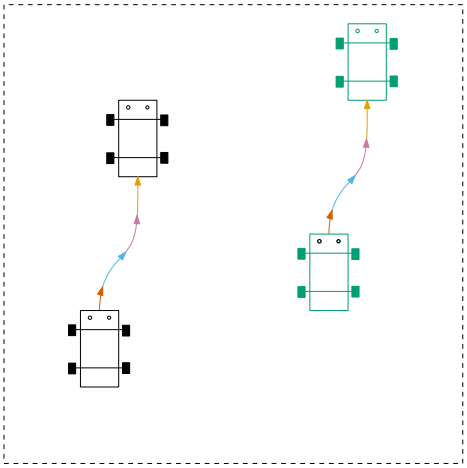
Figure 4.4: Illustration of the integration principle

This second car is an exact copy of the first one. It moves in the same way, provided there is no perturbation as it is the case here. Knowing this, the final position we think about is the one drawn in Figure 4.5a. And with a guaranteed integration scheme we can verify that our hypothesis is valid (Figure 4.5b). However, when looking for an a priori result, your mind probably followed these steps:

1. "Well it is a copy of the first object" (orange arrow in Figure 4.5c)
2. "So it should move the same way" (black arrow in Figure 4.5c)
3. "And thus the result obtained by following the same pattern is this one" (blue arrow in Figure 4.5c)

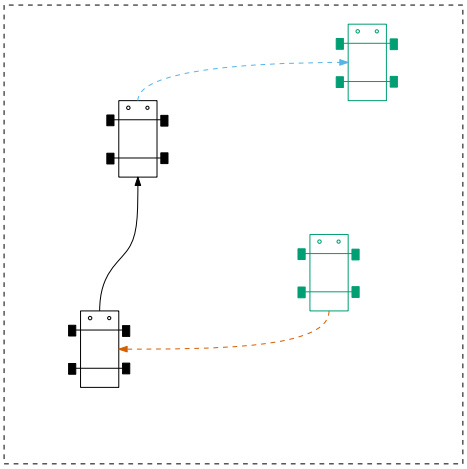


(a) A priori final position in the 2nd case

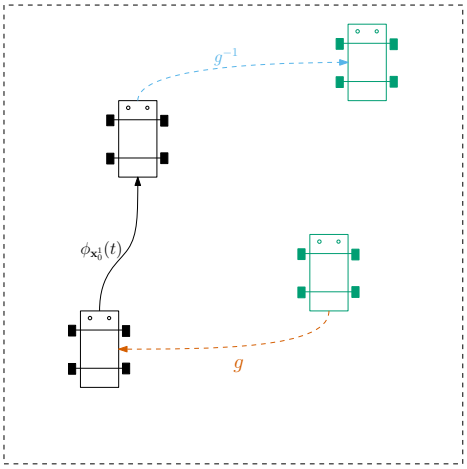


(b) Guaranteed integration steps 2nd case

Figure 4.5: Illustration of the integration principle (continued)



(c) Work of mind



(d) Mathematical representation

Figure 4.5: Illustration of the integration principle (continued)

If one translates this work of mind from english to maths, the result obtained is something like this:

1. There exists a function g such that $g(\mathbf{x}_0^2) = \mathbf{x}_0^1$
2. The image of $\mathbf{x}_0^1$ is given by $\phi_1(\mathbf{x}_0^1)$
3. Using the inverse g^{-1} we can compute $\phi_1(\mathbf{x}_0^2)$

The main interest here, is that provided the image $\phi_1(\mathbf{x}_0^1)$ is known, then computing $\phi_1(\mathbf{x}_0^2)$ is done in three steps instead of 4 (or more if more steps are need when using a classic integration scheme). In this chapter, we will show how such functions g can be found using Lie groups and how to apply this principle to guaranteed integration.

4.3 Lie Symmetries applied to differential equations

4.3.1 Origins

The research on transformation groups applied to differential equations is an old field. It started when the Norwegian mathematician Sophus Lie (from which Lie groups are named) decided to investigate continuous transformation groups leaving differential equations invariant [121]. With his work, a new field called *symmetry analysis for differential equations* appeared. Lie developed it in [80, 81]. Unfortunately, the concept disappeared a bit and only the Lie groups were kept until the middle of the 20^{th} century. It came back with the work of Birkhoff [11] and has been the subject of numerous research since then [12, 96, 97, 108].

4.3.2 Actions and stabilisers

In this section, we will define the different objects and operations we will use to perform the guaranteed integration. Most of them are derived from the definitions presented in Section 4.1, adapted to the context of differential equations. Hence, in the following, the set $\mathbb{A}$ which was previously a set of three elements $\{a, b, c\}$ will now correspond to the state space $\mathbb{R}^n$ in which the state $\mathbf{x}$ of our moving object evolves. Moreover, the set $\mathbb{F}$ now gathers all the differential state equations of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. In addition, we assume that $\mathbf{f}$ is locally Lipschitz in the rest of this thesis so that the flow function $\phi_t : \mathbb{R}^n \to \mathbb{R}^n$ is defined (see Section 2.2.2.2). This flow function associates to all initial vector $\mathbf{x}_0$ the solution $\phi_t(\mathbf{x}_0)$ of the state equation.

Remark 4.2. As seen in Example 2.1, an evolution function can be seen as a vector field. To ease the reader understanding, we will voluntarily use $\mathbf{f}$ to denote both the evolution function and the vector field associated with it.

Remark 4.3. We denote by $\text{diff}(\mathbb{R}^n)$ the set of diffeomorphisms from $\mathbb{R}^n$ to $\mathbb{R}^n$.

We have defined in Section 4.1 the left group action. In the context of differential equations, our group action is defined as follows:

Definition 4.7. (Action of diffeomorphisms) Consider a state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{g} \in \text{diff}(\mathbb{R}^n)$. We define the action of $\mathbf{g}$ on $\mathbf{f}$ as:

$$\mathbf{g} \bullet \mathbf{f} = \left(\frac{d\mathbf{g}}{d\mathbf{x}} \circ \mathbf{g}^{-1} \right) \cdot (\mathbf{f} \circ \mathbf{g}^{-1}). \tag{4.5}$$

Geometrically, this action transforms a vector field $\mathbf{f}$ into another one. To prove that the action defined in Equation (4.5) is a group action we first need to prove that $(\text{diff}(\mathbb{R}^n), \circ)$ is a group. This is done through Proposition 4.1 and Proposition 4.2.

Proposition 4.1. *Assume that* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *and* $\mathbf{y} = \mathbf{g}(\mathbf{x})$ *where* $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{g} \in \text{diff}(\mathbb{R}^n)$. *Then, we have* $\dot{\mathbf{y}} = \mathbf{g} \bullet \mathbf{f}(\mathbf{y})$. *Equivalently, the action of* $\mathbf{g}$ *generates from the system* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *the new system* $\dot{\mathbf{y}} = \mathbf{g} \bullet \mathbf{f}(\mathbf{y})$.

Proof. During a given amount of time dt, infinitely small, $\mathbf{x}(t)$ has moved of $d\mathbf{x} = dt \cdot \mathbf{f}(\mathbf{x}(t))$ as a first order approximation. Therefore we have:

$$\begin{aligned}
\mathbf{y}(t+dt) &= \mathbf{g}(\mathbf{x}(t+dt)) \\
&= \mathbf{g}(\mathbf{x}(t)) + \tfrac{d\mathbf{g}}{d\mathbf{x}}(\mathbf{x}(t)) \cdot \tfrac{d\mathbf{x}}{dt}(t) \cdot dt + \mathbf{o}(dt) \\
&= \mathbf{g}(\mathbf{x}(t)) + \tfrac{d\mathbf{g}}{d\mathbf{x}}(\mathbf{x}(t)) \cdot \mathbf{f}(\mathbf{x}(t)) \cdot dt + \mathbf{o}(dt) \\
&= \mathbf{y}(t) + dt \cdot \tfrac{d\mathbf{g}}{d\mathbf{x}}(\mathbf{g}^{-1}(\mathbf{y}(t))) \cdot \mathbf{f}(\mathbf{g}^{-1}(\mathbf{y}(t))) + \mathbf{o}(dt) \\
&= \mathbf{y}(t) + dt \cdot (\mathbf{g} \bullet \mathbf{f})(\mathbf{y}(t)) + \mathbf{o}(dt).
\end{aligned}$$

Therefore, due to the unicity of the first order Taylor expansion, we get $\dot{\mathbf{y}} = \mathbf{g} \bullet \mathbf{f}(\mathbf{y})$ □

Proposition 4.2. *The structure $G(diff(\mathbb{R}^n), \circ)$ composed of the set of diffeomorphisms from $\mathbb{R}^n$ to $\mathbb{R}^n$ and the composition operator $\circ$ is a group and the action $\bullet$, as defined in Definition 4.7, a left group action of G.*

Proof. We first need to check that G is a group.

1. (closure) If $\mathbf{g}_1, \mathbf{g}_2 \in \text{diff}(\mathbb{R}^n)$ then $\mathbf{g}_1 \circ \mathbf{g}_2 \in \text{diff}(\mathbb{R}^n)$,
2. (associativity) The associativity requirement is satisfied as for all $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \in \text{diff}(\mathbb{R}^n)$, $(\mathbf{g}_1 \circ \mathbf{g}_2) \circ \mathbf{g}_3 = \mathbf{g}_1 \circ (\mathbf{g}_2 \circ \mathbf{g}_3)$,
3. (identity) The identity function is the identity element of $\text{diff}(\mathbb{R}^n)$,
4. (inverse) As for each element $\mathbf{g} \in G$, $\mathbf{g} \in \text{diff}(\mathbb{R}^n)$ by definition it is bijective thus there exists an inverse $\mathbf{g}^{-1} \in G$.

Now let us see if $\bullet$ satisfies the identity and compatibility properties. We consider the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$.

1. (identity) Denote by $\mathbf{i}$ the identity element of G and $\mathbf{p} = \mathbf{i}(\mathbf{x})$. From Proposition 4.1, we have:
$$\begin{aligned}
\dot{\mathbf{p}} &= \mathbf{i} \bullet \mathbf{f}(\mathbf{p}) \\
&= \left(\frac{d\mathbf{i}}{d\mathbf{p}} \circ \mathbf{i}^{-1}\right) \cdot \left(\mathbf{f} \circ \mathbf{i}^{-1}\right)(\mathbf{p}) \\
&= (1 \circ \mathbf{i}) \cdot (\mathbf{f} \circ \mathbf{i})(\mathbf{p}) \\
&= 1 \cdot \mathbf{f}(\mathbf{p}) \\
&= \mathbf{f}(\mathbf{p}).
\end{aligned}$$
Thus $\mathbf{i} \bullet \mathbf{f} = \mathbf{f}$, the identity property is satisfied.
2. (compatibility) We introduce $\mathbf{g}, \mathbf{h} \in G$, $\mathbf{y} = \mathbf{g}(\mathbf{x})$ and $\mathbf{z} = \mathbf{h} \circ \mathbf{g}(\mathbf{x})$.
   (i) On the one hand, from Proposition 4.1 :
   $$\dot{\mathbf{z}} = ((\mathbf{h} \circ \mathbf{g}) \bullet \mathbf{f})(\mathbf{z}),$$
   (ii) On the other hand, since $\mathbf{z} = \mathbf{h}(\mathbf{y})$ and $\dot{\mathbf{y}} = \mathbf{g} \bullet \mathbf{f}(\mathbf{x})$, we have from Proposition 4.1:
   $$\dot{\mathbf{z}} = (\mathbf{h} \bullet (\mathbf{g} \bullet \mathbf{f}))(\mathbf{z}),$$

Therefore

$$(\mathbf{h} \circ \mathbf{g}) \bullet \mathbf{f} = \mathbf{h} \bullet (\mathbf{g} \bullet \mathbf{f}),$$

and the compatibility property is satisfied.

□

Remark 4.4. From now on, we will denote by G the group (diff($\mathbb{R}^n$),$\circ$) (regardless of the value of $n \in \mathbb{N}^*$) and by $\bullet$ the action defined in Definition 4.7.

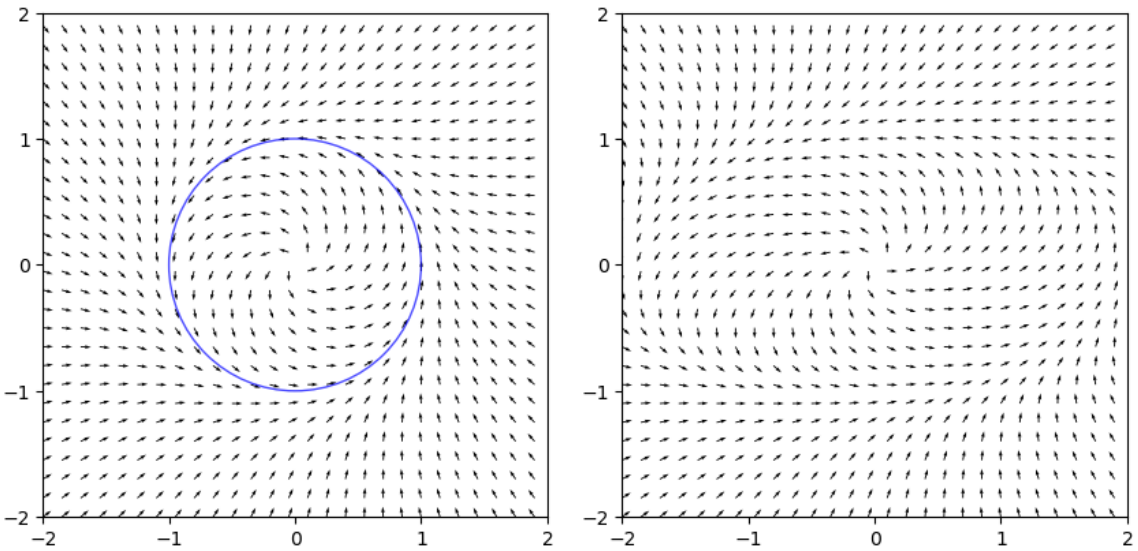
Example 4.7 (Group action on vector field)**.** Consider the dynamical system defined in Example 2.1. We recall its state equation:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} -x_1^3 - x_1 x_2^2 + x_1 - x_2 \\ -x_2^3 - x_1^2 x_2 + x_1 + x_2 \end{pmatrix} \tag{4.6}$$

We introduce $\mathbf{h} \in G$ such that:

$$\mathbf{h} : \begin{array}{ccc} \mathbb{R}^2 & \to & \mathbb{R}^2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \mapsto & \begin{pmatrix} 2x_1 \\ x_2 \end{pmatrix} \end{array}$$

Figure 4.6 illustrates the effect of the action of $\mathbf{h}$ on $\mathbf{f}$.



(a) $\mathbf{f}$ (b) $\mathbf{h} \bullet \mathbf{f}$

Figure 4.6: Effect of the action of $\mathbf{h}$ on $\mathbf{f}$. The vector field $\mathbf{f}$ is turned into another vector field (here $\mathbf{h}$ is a scaling operation).

As seen in Definition 4.6, a transformation $\mathbf{g} \in G$ is a stabiliser of $\mathbf{f}$ if $\mathbf{g} \bullet \mathbf{f} = \mathbf{f}$. In our current case $\mathbf{g}$ is a stabiliser if it satisfies the partial differential equation

$$\mathbf{g} \bullet \mathbf{f} = \left(\frac{d\mathbf{g}}{d\mathbf{x}} \circ \mathbf{g}^{-1} \right) \cdot (\mathbf{f} \circ \mathbf{g}^{-1}) = \mathbf{f}. \tag{4.7}$$

We recall that the subset of stabilisers of $\mathbf{f}$ included in $\text{diff}(\mathbb{R}^n)$ is a subgroup of G (see Definition 4.6) with the composition rule. This subgroup is denoted $Sym_G(\mathbf{f})$.

Remark 4.5. Equation (4.7) is equivalent to

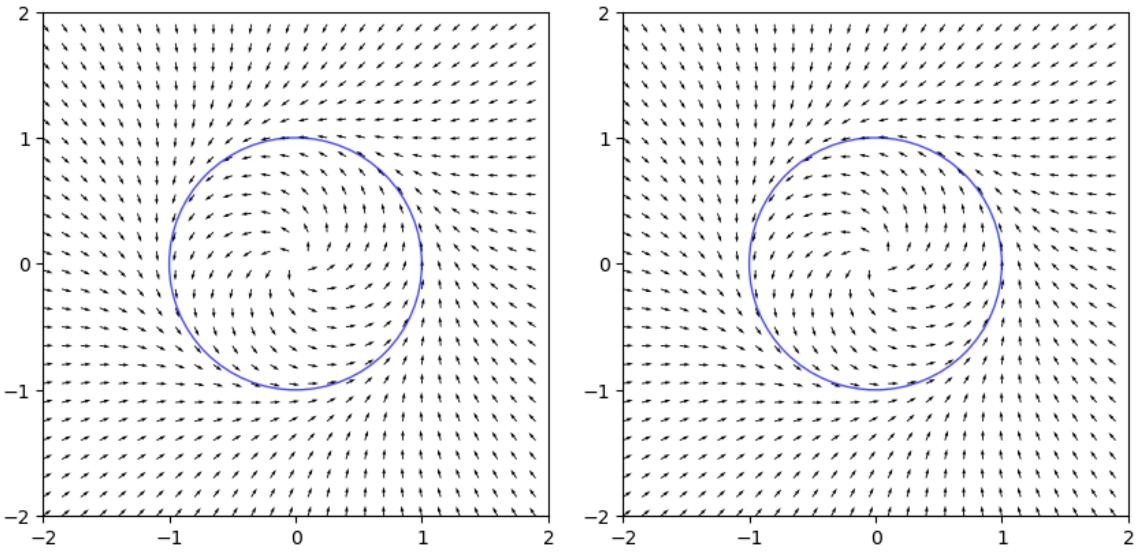
$$\left(\frac{d\mathbf{g}}{d\mathbf{x}}\right) \cdot \mathbf{f} = \mathbf{f} \circ \mathbf{g}. \tag{4.8}$$

When $\mathbf{g}$ is linear, we get $\mathbf{g} \circ \mathbf{f} = \mathbf{f} \circ \mathbf{g}$. This means that both $\mathbf{f}$ and $\mathbf{g}$ commute by composition. This is known as the *equivariance* property [45].

Example 4.8. (Stabiliser on vector field) Once again we consider the dynamical system defined in Example 4.7. This time we introduce $\mathbf{r} \in G$ such that

$$\begin{array}{rccc} & \mathbb{R}^2 & \to & \mathbb{R}^2 \\ \mathbf{r}: & \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \mapsto & \begin{pmatrix} cos\left(\frac{\pi}{4}\right) x_1 - sin\left(\frac{\pi}{4}\right) x_2 \\ cos\left(\frac{\pi}{4}\right) x_2 + sin\left(\frac{\pi}{4}\right) x_1 \end{pmatrix} \end{array}$$

which corresponds to a rotation of $\frac{\pi}{4}$. The action of $\mathbf{r}$ on $\mathbf{f}$ is depicted in Figure 4.7.



(a) $\mathbf{f}$ (b) $\mathbf{r} \bullet \mathbf{f}$

Figure 4.7: Effect of the action of $\mathbf{r}$ on $\mathbf{f}$.

As one can notice in Figure 4.7, the action of $\mathbf{r}$ on $\mathbf{f}$ leaves the vector field unchanged. Therefore $\mathbf{r}$ is a stabiliser of $\mathbf{f}$.

The stabilisers defined above will be a key tool for our new guaranteed integration method. However, if we know which properties must be satisfied, finding the corresponding stabilisers is not always easy. The three following properties will show how to find stabilisers based on the knowledge of only one.

Proposition 4.3. *If $\phi_t : \mathbb{R}^n \to \mathbb{R}^n$ is the flow associated with the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, then we have*

$$\mathbf{g} \bullet \mathbf{f} = \mathbf{f} \iff \phi_t \circ \mathbf{g} = \mathbf{g} \circ \phi_t. \tag{4.9}$$

Proof. Suppose $\mathbf{y} = \mathbf{g}(\mathbf{x})$ and $\mathbf{x}_0 \in \mathbb{R}^n$. For the corresponding trajectory $\mathbf{x}(t) = \phi_t(\mathbf{x}_0)$, we have:

$$\begin{aligned}
\mathbf{g} \circ \phi_t(x_0) = \phi_t \circ \mathbf{g}(x_0), \forall t &\iff \mathbf{g}(\mathbf{x}(t)) = \phi_t(\mathbf{g}(\mathbf{x}_0)), \forall t \\
&\iff \mathbf{y}(t) = \phi_t(\mathbf{y}(0)), \forall t \\
&\iff \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \\
&\iff \mathbf{g} \bullet \mathbf{f} = \mathbf{f}.
\end{aligned}$$

The last equivalence comes from Proposition 4.1 which states that $\dot{\mathbf{y}} = \mathbf{g} \bullet \mathbf{f}(\mathbf{y})$. On the one hand we have :

$$\begin{cases} \dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}) \\ \dot{\mathbf{y}} &= \mathbf{g} \bullet \mathbf{f}(\mathbf{y}) \end{cases} \implies \mathbf{g} \bullet \mathbf{f} = \mathbf{f}.$$

And on the other hand:

$$\begin{cases} \mathbf{g} \bullet \mathbf{f} &= \mathbf{f} \\ \dot{\mathbf{y}} &= \mathbf{g} \bullet \mathbf{f}(\mathbf{y}) \end{cases} \implies \dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}).$$

□

The last proposition brings a vital property for the rest of this work. If we consider the known solution $t \mapsto \phi_t(\mathbf{x}_0)$ of the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, for $\mathbf{x}(0) = \mathbf{x}_0$, and setting $\mathbf{x}_1 = \mathbf{g}(\mathbf{x}_0)$, where $\mathbf{g}$ is a stabiliser of $\mathbf{f}$, we get from Proposition 4.3:

$$\phi_t(\mathbf{x}_1) = \phi_t \circ \mathbf{g}(\mathbf{x}_0) \overset{(4.9)}{=} \mathbf{g} \circ \phi_t(\mathbf{x}_0). \tag{4.10}$$

One can notice that we get an expression of the solution of the state equation corresponding to the initial condition $\mathbf{x}(0) = \mathbf{x}_1$ depending on the known solution for the initial condition for $\mathbf{x}(0) = \mathbf{x}_0$. Hence, if we have a family of stabilisers, it is then possible to **generate a family of solutions from the unique known solution** $\phi_t(\mathbf{x}_0)$.

Example 4.9. (One parameter flow symmetry) Consider $t_1 \in \mathbb{R}$. Since we have

$$\phi_t \circ \phi_{t_1} = \phi_{t_1} \circ \phi_t = \phi_{t+t_1}, \tag{4.11}$$

from Proposition 4.3 we have $\phi_{t_1} \bullet \mathbf{f} = \mathbf{f}$, therefore ϕ_{t_1} is a stabiliser of $\mathbf{f}$. The function ϕ_{t_1} is called the *one-parameter flow symmetry*

Proposition 4.4. *Consider a state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ and $\mathbf{g}$ a stabiliser of $\mathbf{f}$. In a new coordinate system $\mathbf{y} = \mathbf{h}(\mathbf{x})$, where $\mathbf{h}$ is bijective and smooth, the action $\mathbf{h} \circ \mathbf{g} \circ \mathbf{h}^{-1}$ is a stabiliser.*

Proof. In the $\mathbf{y}$ space, the flow is defined by:

$$\psi_t(\mathbf{y}) = \mathbf{h} \circ \phi_t \circ \mathbf{h}^{-1}(\mathbf{y}). \tag{4.12}$$

Since $\mathbf{g}$ is a stabiliser for the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, we have:

$$\begin{aligned}
\phi_t(\mathbf{x}) = \mathbf{g} \circ \phi_t \circ \mathbf{g}^{-1}(\mathbf{x}) &\iff \phi_t \circ \mathbf{h}^{-1}(\mathbf{y}) = \mathbf{g} \circ \phi_t \circ \mathbf{g}^{-1} \circ \mathbf{h}^{-1}(\mathbf{y}) \\
&\iff \underbrace{\mathbf{h} \circ \phi_t \circ \mathbf{h}^{-1}(\mathbf{y})}_{\psi_t(\mathbf{y})} = \mathbf{h} \circ \mathbf{g} \circ \underbrace{\phi_t}_{\mathbf{h}^{-1} \circ \psi_t \circ \mathbf{h}} \circ \mathbf{g}^{-1} \circ \mathbf{h}^{-1}(\mathbf{y}) \\
&\iff \psi_t(\mathbf{y}) = (\mathbf{h} \circ \mathbf{g} \circ \mathbf{h}^{-1}) \circ \psi_t \circ (\mathbf{h} \circ \mathbf{g}^{-1} \circ \mathbf{h}^{-1})(\mathbf{y}) \\
&\iff \psi_t \circ (\mathbf{h} \circ \mathbf{g}^{-1} \circ \mathbf{h}^{-1})(\mathbf{y}) = (\mathbf{h} \circ \mathbf{g} \circ \mathbf{h}^{-1}) \circ \psi_t(\mathbf{y}).
\end{aligned}$$

From the last equivalence and Proposition 4.3, $\mathbf{h} \circ \mathbf{g}^{-1} \circ \mathbf{h}^{-1}$ is a stabiliser for the system in the $\mathbf{y}$ coordinates. □

4.4 Transport function

This section is dedicated to a new tool that we will call transport function. It will be a key element of the new method of integration developed in the next chapter.

4.4.1 Moving between trajectories

As mentioned in the previous section, stabilisers will be the ground on which our method will be built. Let us introduce the first of the four test-cases, Test-Case 1 (TC1), that will be studied throughout this work. We consider the system:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} 1 \\ -x_2 \end{pmatrix} \tag{4.13}$$

Its vector field is depicted in Figure 4.8. As it can be easily seen in the figure, the mirror-symmetry w.r.t the abscissa axis Ox_1 is a stabiliser for the system as the vector field does not change. Let us denote it by $\mathbf{g}_1$. Moreover, the horizontal translation along the abscissa, denoted $\mathbf{g}_2$ is also a stabiliser. Finally, coming from Proposition 4.3, and seen in Example 4.9, for all $t_1 \in \mathbb{R}$, we have

$$\phi_t \circ \phi_{t_1} = \phi_{t_1} \circ \phi_t$$

thus ϕ_{t_1} is a stabiliser. Whence, $\mathbf{g}_1$, $\mathbf{g}_2$, ϕ_{t_1} all belong to $\mathrm{Sym}(\mathbf{f})$.

In Figure 4.8, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{h}$ are fixed points in the state space to illustrate how it is possible to move along the trajectories using stabilisers. On the figure, the trajectory painted in red on which $\mathbf{b}$ and $\mathbf{c}$ are located is one solution we know. We call it *the reference*. All the other are not known to us. Let us compute $\phi_t(\mathbf{a})$. We have:

$$\phi_t(\mathbf{a}) = \mathbf{g}_1 \circ \phi_t \circ \mathbf{g}_1^{-1}(\mathbf{a}) = \mathbf{d}.$$

This means that we first move from $\mathbf{a}$ to $\mathbf{b}$ (change in the coordinate system) following $\mathbf{g}^{-1}$. Then as we are on the *reference*, we get $\mathbf{c} = \phi_t(\mathbf{b})$. Finally we move back to our initial coordinate system using $\mathbf{g}$ to get $\mathbf{d} = \mathbf{g}(\mathbf{c}) = \phi_t(\mathbf{a})$. In the same manner, since $\phi_t(\mathbf{h}) = \mathbf{g}_2 \circ \phi_t \circ \mathbf{g}_2^{-1}(\mathbf{h})$, we can compute $\mathbf{e} = \phi_t(\mathbf{h})$, using the knowledge we have of the reference and the symmetry $\mathbf{g}_2$. If we assume that we can compute accurately the solution ϕ_t, called the reference, for a specific initial state, then we will be able to move precisely forward and backward in time from every reachable point with the symmetries from the reference.

4.4.2 Lie group of symmetries

To move between the different trajectories as seen in Section 4.4.1, we need to find symmetries. And to do so from everywhere in the state space, we need a group of them so we

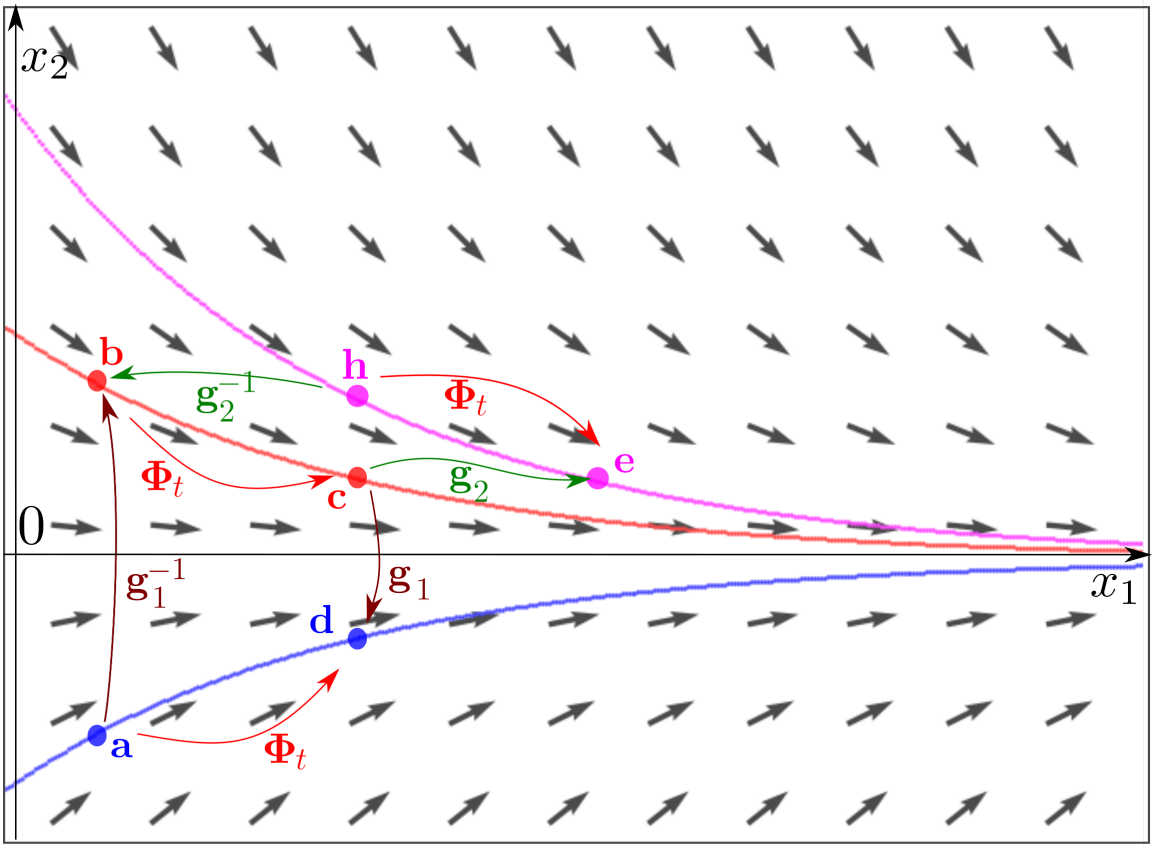


Figure 4.8: Test-case 1: The system has an x_1 translation symmetry and an Ox_1 mirror-symmetry.

can link any point of the state space to the reference. Moreover, we need complementary symmetries, *i.e* symmetries that make us move along each axis of the state space. In TC1, they are Ox_1 and Ox_2. In the context of differential equations, these groups are called *Lie Groups of symmetries*.

Definition 4.8 (Lie group of symmetries)**.** Consider a state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ and an manifold $\mathbb{P}$. A Lie group $G_{\mathbf{p}}$ of symmetries is a family of diffeomorphisms $\mathbf{g}_{\mathbf{p}} \in \mathit{diff}(\mathbb{R}^n)$ parameterised by $\mathbf{p} \in \mathbb{P}$ such that:

- $G_{\mathbf{p}}$ is a Lie group with respect to the composition $\circ$,
- $\forall \mathbf{p} \in \mathbb{P}, \mathbf{g}_{\mathbf{p}} \bullet \mathbf{f} = \mathbf{f}$.

The different Lie Symmetries are usually found from the physics and intuition of the system. For instance, from the vector field $\mathbf{f}$ presented in Example 4.8, we can feel that any rotation by an angle θ will yield an identical vector field. Thus, the rotations will be stabilisers of the state equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. Nevertheless, for many systems such as the chaotic Lorenz attractor, such symmetries probably do not exist [117]. For such chaotic systems, there is probably no diffeomorphism that transforms one trajectory into another.

Example 4.10 (Lie group of symmetries)**.** Let us consider the system of TC1. To be able to move everywhere in the state space we need to find at least one group of symmetry to move along Ox_1 and one for Ox_2. We noticed that we have a translation symmetry along Ox_1. This can be written as

$$\mathbf{g}_\alpha : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 + \alpha \\ x_2 \end{pmatrix} \tag{4.14}$$

with $\alpha \in \mathbb{R}$. Let us show that $\mathbf{g}_\alpha$ is a stabiliser for any $\alpha \in \mathbb{R}$.

$$\begin{aligned}
\mathbf{g}_\alpha \bullet \mathbf{f}(\mathbf{x}) &= \left(\frac{d\mathbf{g}_\alpha}{d\mathbf{x}} \circ \mathbf{g}_\alpha^{-1} \right) \cdot \left(\mathbf{f} \circ \mathbf{g}_\alpha^{-1} \right)(\mathbf{x}) \\
&= \left(\frac{d\mathbf{g}_\alpha}{d\mathbf{x}} \cdot \mathbf{f} \right) \circ \mathbf{g}_\alpha^{-1}(\mathbf{x}) \\
&= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -x_2 \end{pmatrix} \right) \circ \begin{pmatrix} x_1 - \alpha \\ x_2 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ -x_2 \end{pmatrix} \\
&= \mathbf{f}(\mathbf{x})
\end{aligned}$$

Therefore $\mathbf{g}_\alpha$ is a stabiliser of $\mathbf{f}$ for any $\alpha \in \mathbb{R}$. Secondly, we have a mirror symmetry w.r.t the Ox_1 axis. This will be written as function $\mathbf{g}_\beta$ such that

$$\mathbf{g}_\beta : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ \beta x_2 \end{pmatrix}. \tag{4.15}$$

Then again, let us prove that $\mathbf{g}_\beta$ is a stabiliser of $\mathbf{f}$ for any $\beta \in \mathbb{R}$.

$$\begin{aligned}
\mathbf{g}_\beta \bullet \mathbf{f}(\mathbf{x}) &= \left(\frac{d\mathbf{g}_\beta}{d\mathbf{x}} \circ \mathbf{g}_\beta^{-1} \right) \cdot \left(\mathbf{f} \circ \mathbf{g}_\beta^{-1} \right)(\mathbf{x}) \\
&= \left(\frac{d\mathbf{g}_\beta}{d\mathbf{x}} \cdot \mathbf{f} \right) \circ \mathbf{g}_\beta^{-1}(\mathbf{x}) \\
&= \left(\begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -x_2 \end{pmatrix} \right) \circ \begin{pmatrix} x_1 \\ \frac{x_2}{\beta} \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ -x_2 \end{pmatrix} \\
&= \mathbf{f}(\mathbf{x})
\end{aligned}$$

Hence $\mathbf{g}_\beta$ is a stabiliser of $\mathbf{f}$ for any $\beta \in \mathbb{R}$. Now let us define $\mathbf{g}_\mathbf{p}$ with $\mathbf{p} \in \mathbb{R}^2$ such that:

$$\mathbf{g}_\mathbf{p} : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 + p_1 \\ p_2 x_2 \end{pmatrix}. \tag{4.16}$$

We have already proven that $\mathbf{g}_\mathbf{p}$ is a stabiliser of $\mathbf{f}$ whatever the value of $\mathbf{p}$. Indeed for any value of $\mathbf{p}$,

$$\mathbf{g}_\mathbf{p} = \mathbf{g}_\alpha \circ \mathbf{g}_\beta = \mathbf{g}_\beta \circ \mathbf{g}_\alpha, \tag{4.17}$$

with $\alpha, \beta \in \mathbb{R}$. As $\mathbf{g}_\alpha$ and $\mathbf{g}_\beta \in \mathrm{Sym}(\mathbf{f})$ and $\mathrm{Sym}(\mathbf{f})$ is a group by composition rule, then $\mathbf{g}_\mathbf{p} \in \mathrm{Sym}(\mathbf{f})$ (closure property).

Let us denote by $G_{\mathbf{p}}$ the group composed of the set of diffeomorphisms $\mathbf{g_p}$ as defined in Equation (4.16) with $\mathbf{p} \in \mathbb{R}^2$ and the composition rule. $G_{\mathbf{p}}$ is a Lie group w.r.t the composition rule. In addition, for all $\mathbf{p} \in \mathbb{R}^2, \mathbf{g_p} \bullet \mathbf{f} = \mathbf{f}$. Thus $G_{\mathbf{p}}$ is a lie group of symmetries.

4.4.3 The transport function

This section will introduce a tool which will be fundamental in the rest of this work. We call it *the transport function*. Consider a Lie group of symmetries $G_{\mathbf{p}}$ as presented in the previous section. We assume that $G_{\mathbf{p}}$ is transitive, *i.e* it has only one orbit (see Definition 4.5). In this case, there is a function $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{P}$ such that $\mathbf{h}(\mathbf{x}, \mathbf{a})$ corresponds to the displacement $\mathbf{p}$ to be chosen so that the point $\mathbf{a}$ is moved to $\mathbf{x}$ by $\mathbf{g_p}$, which means:

$$\mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a})}(\mathbf{a}) = \mathbf{x}. \tag{4.18}$$

This new transport function object is related to what is called the *moving frame method*. For more information on it, the reader is advised to read [13]. The main difference from what is presented in [13] is the fact that we assume that there is only one orbit which allows us to avoid the introduction of cross-section (see [119]).

Remark 4.6. The transport function is not necessary unique.

Remark 4.7. From now on, the function $\mathbf{h}$ will always represent the transport function. Moreover, for the rest of this thesis, $\mathbf{a}$ (or $a, a(\cdot), \mathbf{a}(\cdot), [a], [\mathbf{a}], [a](\cdot), [\mathbf{a}](\cdot)$) will represent the *reference*, may it be a vector, an interval, a tube ...

Example 4.11 (Transport function)**.** Let us introduce the second test-case, Test-Case 2 (TC2). The system considered follows the state equation:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix}. \tag{4.19}$$

Its associated vector field is given in Figure 4.9. From this vector field, we can see that there is a translation symmetry along the Ox_2 axis. Furthermore, if we perform a translation of $p_2 = \pm k2\pi$ along Ox_1, the field does not change either. However, p_2 belongs to a discrete set, which is not compatible with a Lie symmetry as it requires a continuous manifold. We thus introduce the transformations $\mathbf{g_p}$ such that

$$\mathbf{g_p}(\mathbf{x}) = \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \phi_{p_2}(\mathbf{x}). \tag{4.20}$$

Now let us prove that these transformations are Lie symmetries for the system of TC2.

Proof. If we define

$$\mathbf{g}^1_{p_1} : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ p_1 + x_2 \end{pmatrix}, p_1 \in \mathbb{R}, \tag{4.21}$$

and

$$\mathbf{g}^2_{p_2} : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \phi_{p_2}(\mathbf{x}), p_2 \in \mathbb{R}, \tag{4.22}$$

then we have $\mathbf{g_p} = \mathbf{g}^1_{p_1} \circ \mathbf{g}^2_{p_2}$. If we manage to check that both are stabilisers then $\mathbf{g_p}$ will be a stabiliser.

- We know that $\mathbf{g}^1_{p_1}$ is linear and we have

$$\left(\frac{d\mathbf{g}^1_{p_1}}{d\mathbf{x}} \cdot \mathbf{f}\right)(\mathbf{x}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix} = \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix}, \tag{4.23}$$

  and

$$\mathbf{f} \circ \mathbf{g}^1_{p_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix} \circ \begin{pmatrix} x_1 \\ p_1 + x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix}, \tag{4.24}$$

  Thus

$$\left(\frac{d\mathbf{g}^1_{p_1}}{d\mathbf{x}}\right) \cdot \mathbf{f} = \mathbf{f} \circ \mathbf{g}^1_{p_1}.$$

  From the equivariance property seen in Remark 4.5, $\mathbf{g}^1_{p_1}$ is a stabiliser.

- Secondly, we have seen in Example 4.9 that ϕ_{p_2} is a stabiliser for all $p_2 \in \mathbb{R}$. Thus $\mathbf{g}^2_{p_2}$ is a stabiliser.

Hence the transformations $\mathbf{g_p}$ with $\mathbf{p} \in \mathbb{R}^2$ as described in Equation (4.20) are stabilisers of the system considered in TC2. □

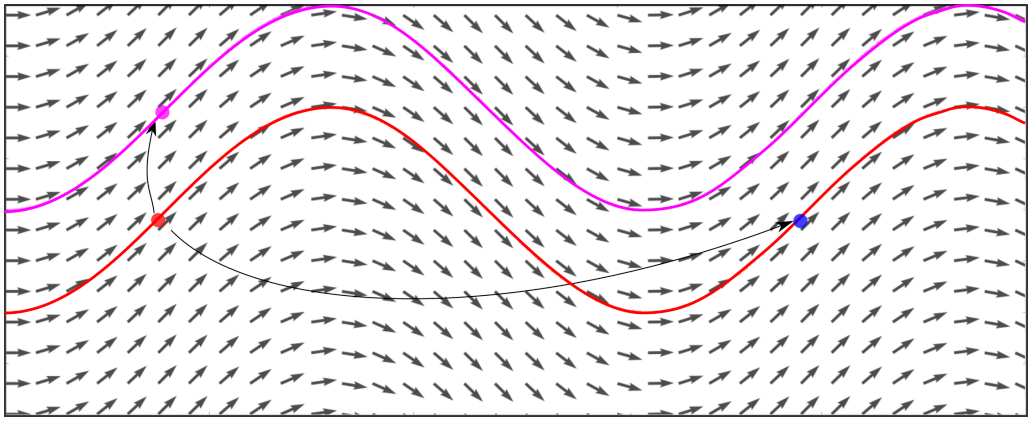


Figure 4.9: Test-case 2 vector field. It is possible to move from the red dot on the red curve to the magenta dot on the magenta curve using a translation along Ox_2. We can also move from the red dot to the blue dot using a $\pm k2\pi$ translation along Ox_2.

Now let us find a transport function $\mathbf{h}$ associated with these symmetries.

$$\begin{aligned}
\mathbf{g_p}(\mathbf{a}) = \mathbf{x} &\iff \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \phi_{p_2}(\mathbf{a}) = \mathbf{x} \\
&\iff \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \begin{pmatrix} \phi_{p_2,1}(\mathbf{a}) \\ \phi_{p_2,2}(\mathbf{a}) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\
&\iff \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \begin{pmatrix} a_1 + p_2 \\ \phi_{p_2,2}(\mathbf{a}) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\
&\iff \begin{pmatrix} p_2 \\ p_1 + \phi_{x_1-a_1,2}(\mathbf{a}) \end{pmatrix} = \begin{pmatrix} x_1 - a_1 \\ x_2 \end{pmatrix} \\
&\iff \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \underbrace{\begin{pmatrix} x_2 - \phi_{x_1-a_1,2}(\mathbf{a}) \\ x_1 - a_1 \end{pmatrix}}_{\mathbf{h}(\mathbf{x},\mathbf{a})}.
\end{aligned}$$

Therefore, one possible transport function is

$$\mathbf{h}(\mathbf{x}, \mathbf{a}) = \begin{pmatrix} x_2 - \phi_{x_1-a_1,2}(\mathbf{a}) \\ x_1 - a_1 \end{pmatrix}. \tag{4.25}$$

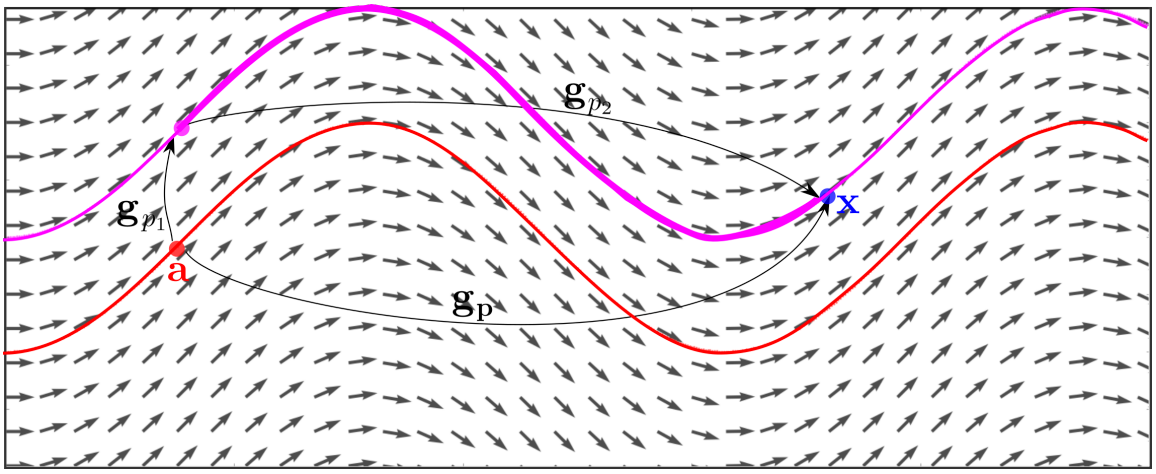


Figure 4.10: Using the transport function we can find the value of $\mathbf{p}$ which sends $\mathbf{a}$ on $\mathbf{x}$ by using $\mathbf{g_p}$

Example 4.12 (Transport function TC1)**.** To ease the reader understanding, we will repeat the process and try to find a transport function using the symmetries of TC1. We ended with functions of the form

$$\mathbf{g_p} : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 + p_1 \\ p_2 x_2 \end{pmatrix}. \tag{4.26}$$

Applying the same principle as in Example 4.11, we have:

$$\begin{aligned}
\mathbf{g_p}(\mathbf{a}) = \mathbf{x} &\iff \begin{pmatrix} a_1 + p_1 \\ p_2 a_2 \end{pmatrix} = \mathbf{x} \\
&\iff \mathbf{p} = \begin{pmatrix} x_1 - a_1 \\ \frac{x_2}{a_2} \end{pmatrix}.
\end{aligned}$$

Therefore, one possible transport function is

$$\mathbf{h}(\mathbf{x}, \mathbf{a}) = \begin{pmatrix} x_1 - a_1 \\ \frac{x_2}{a_2} \end{pmatrix}. \tag{4.27}$$

Remark 4.8. We will show in the next chapter how this tool has a fundamental role in the combination of Lie groups applied to differential equations with interval methods for guaranteed integration. In what follows, we will assume that we have a closed form for $\mathbf{h}(\mathbf{x}, \mathbf{a})$. However, in practice, it is derived from the symmetries of the problem as shown in Example 4.11 and Example 4.12.

CHAPTER 5

A NEW GUARANTEED INTEGRATION METHOD

Contents

This chapter is dedicated to the main contribution of this thesis, the development of a new guaranteed integration method. We present here how an inclusion of the flow function is found using the mathematical tools presented in the three previous chapters. Then the different steps of the integration are explained in Section 5.2 and illustrated through TC1 and TC2. Comparisons of the results with existing methods will be presented in Section 5.4. A more complicated test case will then be treated in Section 5.5 to show the efficiency of the method. Lastly, we will apply our method on a robot model to show its practicality in this context.

Remark 5.1. One purpose of this work is to encourage the use of the method presented in this chapter. We think that it can be of great help in the robotic field. Thus we will, once again, present and explain some pieces of C++ code. These can be downloaded here for the reader to try (with the installation guidelines). However, if the reader is more acquainted with Python, a python version of the examples is available. Moreover, if one only wants to "see" the computations online without the need to install anything some repls have been prepared for test cases 1, 2, 3 and 4. However the time needed to compute them can be very long due to the use of Python and the server response.

Remark 5.2. All computation have been done with a computer equipped with an Intel I7 8650U@1.90GHz to compare the time needed to compute results between the new method and existing ones.

5.1 Towards a new integration method

5.1.1 Guaranteed integration with an uncertain initial vector: a set inversion problem

As we have seen in Chapter 3, performing a guaranteed integration with an initial condition $\mathbf{x}_0$ comes down to enclosing an evaluation of the flow function $\mathbf{\Phi}_t(\mathbf{x})$ where $\mathbf{\Phi}_0(\mathbf{x}) = \mathbf{x}_0$. Now consider a box $[\mathbf{x}_0]$ which is known to enclose the initial vector $\mathbf{x}_0$. Let us denote by $\mathbb{X}_t$ the set of all states $\mathbf{x}_t$ such that:

$$\begin{cases} \mathbf{\Phi}_t(\mathbf{x}) = \mathbf{x}_t \\ \mathbf{\Phi}_0(\mathbf{x}) = \mathbf{x}_0 \in [\mathbf{x}_0] \end{cases} . \tag{5.1}$$

We want to characterise the sets $\mathbb{X}_t$ for $t \in T$. T can be a continuous interval $T = [0, t_{max}]$, or a discrete set $T = \{t_1, t_2, \ldots, t_m\}$.

Proposition 5.1. *If $\mathbf{\Phi}_t$ is the flow associated with $\mathbf{f}$, then we have*

$$\mathbb{X}_t = \mathbf{\Phi}_{-t}^{-1}([\mathbf{x}_0]). \tag{5.2}$$

Proof. We have:

$$\begin{aligned} \mathbf{x} \in \mathbb{X}_t &\iff \exists \mathbf{x}_0 \in [\mathbf{x}_0], \mathbf{x} = \mathbf{\Phi}_t(\mathbf{x}_0) \\ &\iff \exists \mathbf{x}_0 \in [\mathbf{x}_0], \mathbf{x}_0 = \mathbf{\Phi}_{-t}(\mathbf{x}) \\ &\iff \mathbf{\Phi}_{-t}(\mathbf{x}) \in [\mathbf{x}_0] \\ &\iff \mathbf{x} \in \mathbf{\Phi}_{-t}^{-1}([\mathbf{x}_0]) \end{aligned}$$

Hence

$$\mathbb{X}_t = \mathbf{\Phi}_{-t}^{-1}([\mathbf{x}_0]).$$

□

The consequence of Proposition 5.1 is that characterising $\mathbb{X}_t$ is a set inversion problem. If an inclusion function for $\mathbf{\Phi}_{-t}$ is available for a given t, it is possible to compute an inner and an outer approximation for the different sets $\mathbb{X}_t$ with $t \in T$. This can be done using an algorithm such as SIVIA as presented in Section 2.3.8. As mentioned in Remark 2.16, the computation of these inner and outer approximations is of great importance but also a complicated task, especially for the inner approximation. Some methods have been developed to do so [48], but this new formulation allows us to compute it in a much simpler way.

5.1.2 An inclusion function for the flow

We have just shown that performing a guaranteed integration was equivalent to solving a set inversion problem. If it makes the use of algorithm such as SIVIA possible to solve it we still need a tool, an inclusion function for the flow. But, as seen since the beginning of this work, this flow function is not available in general . Hence the development of the tools seen in Chapter 3. The transport function presented in Section 4.4.3, will be of use.

Let us suppose we have, thanks to the tools presented in Chapter 3, an accurate and guaranteed enclosure $[\mathbf{a}](t)$ for a reference $\mathbf{a}(t) = \mathbf{\Phi}_t(\mathbf{a}_0)$, with $\mathbf{a}(0) = \mathbf{a}_0$. This reference satisfies the equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, as it is the one used to get $[\mathbf{a}](t)$ in the first place. From this reference, let us show that it is possible to find an inclusion function for $\mathbf{\Phi}_t(\mathbf{x})$.

Proposition 5.2. *If* $\mathbf{h}(\mathbf{x}, \mathbf{a})$ *is a transport function for* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$*, then we have:*

$$\mathbf{\Phi}_t(\mathbf{x}) = \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a}_0)} \circ \mathbf{a}(t), \tag{5.3}$$

where $\mathbf{g}_\mathbf{p}$ *is a symmetry of the system.*

Proof. Since $\mathbf{g}_\mathbf{p}$ is a symmetry, we ave for all $\mathbf{p} \in \mathbb{P}$,

$$\mathbf{\Phi}_t(\mathbf{x}) = \mathbf{g}_\mathbf{p} \circ \mathbf{\Phi}_t \circ {\mathbf{g}_\mathbf{p}}^{-1}(\mathbf{x}). \tag{5.4}$$

Taking $\mathbf{p} = \mathbf{h}(\mathbf{x}, \mathbf{a})$, we get

$$\begin{aligned}
\mathbf{\Phi}_t(\mathbf{x}) &= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a_0})} \circ \mathbf{\Phi}_t \circ \mathbf{g}^{-1}_{\mathbf{h}(\mathbf{x},\mathbf{a}_0)}(\mathbf{x}) \\
&= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a_0})} \circ \mathbf{\Phi}_t(\mathbf{a}_0) \text{ (from the transport function, see Equation (4.18))} \\
&= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a_0})} \circ \mathbf{a}(t).
\end{aligned}$$

□

Hence, provided that the reference is known, we have an analytical expression for the flow.

Corollary 5.1. *An inclusion function for* $\mathbf{\Phi}_t(\mathbf{x})$ *is thus*

$$[\mathbf{\Phi}]_{[t]}([\mathbf{x}]) = [\mathbf{g}]_{[\mathbf{h}]([\mathbf{x}],\mathbf{a}_0)}([\mathbf{a}]([t])), \tag{5.5}$$

where $[\mathbf{g}], [\mathbf{h}]$ *are inclusion functions for* $\mathbf{g}, \mathbf{h}$ *and* $[\mathbf{a}](t)$ *encloses the reference* $\mathbf{a}(t)$*. In addition the point* $\mathbf{a}_0$ *is exactly known.*

Proof. Equation (5.5) is obtained by applying the fundamental theorem of interval arithmetic [90] on Equation (5.4) □

This inclusion function is illustrated in Figure 5.1. It computes a box $[\mathbf{y}]$ enclosing $\mathbf{y} = \mathbf{\Phi}_t(\mathbf{x}), \mathbf{x} \in [\mathbf{x}], t \in [t]$. We recall that a closed form expression is available for $\mathbf{g_p}$ and $\mathbf{h}$. We also have a thin enclosure of the reference $[\mathbf{a}](\cdot)$ under the form of a tube.

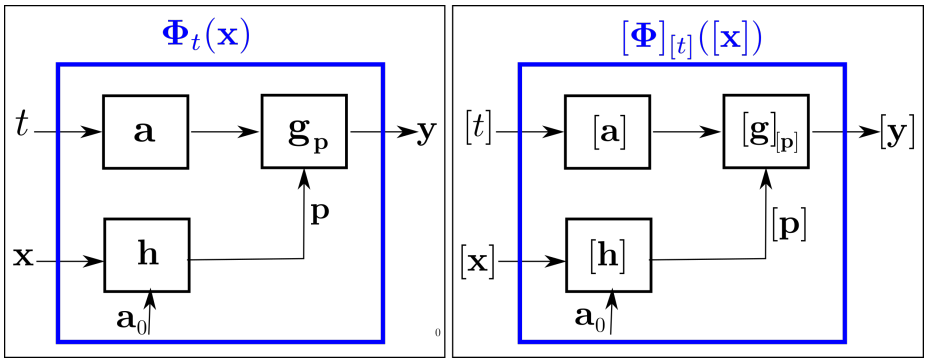


Figure 5.1: Graphical representation of the flow function on the left and its inclusion counterpart on the right

5.2 Integration method

With an inclusion function for the flow function, we now have every tools we need to solve the set inversion problem presented in Section 5.1.1. To do so, we propose the following method:

Step 1: Define a reference $\mathbf{a}(\cdot)$ and enclose it in a tube $[\mathbf{a}](\cdot)$. The initial vector $\mathbf{a}_0$ is perfectly known.

Step 2: Find a Lie group of symmetries $G_{\mathbf{p}}$ and give an expression for the transport function $\mathbf{h}(\mathbf{x}, \mathbf{a})$.

Step 3: Solve the set inversion problem of Proposition 5.1 with the SIVIA algorithm using the inclusion function found in Equation (5.5).

To perform Step 1, we will either use the analytical expression of $\mathbf{a}(t)$ if there is one available to create the tube enclosing the trajectory using Codac as done in Listing 2.1.

Otherwise we will use a guaranteed integration method like one presented in Chapter 3. The group of symmetries and transport functions are for now found by hand using the intuition we have on the system. To implement our integration method, we will use Codac as it gathers all the tools we need. Now let us illustrate the method on several examples.

5.2.1 Integration of a single set at a finite time

Integration schemes first aim at finding a solution at a finite time. For instance, we may want to compute the position of an object after some time $t_{elapsed}$ knowing its evolution function. But what is happening in between time $t = 0$ and $t = t_{elapsed}$ may not be of interest. This particular case is presented in Example 5.1.

Example 5.1 (test case 1, guaranteed integration)**.** Let us come back to TC1. We recall that the system is defined by

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} 1 \\ -x_2 \end{pmatrix}. \tag{5.6}$$

We would like to characterise $\mathbb{X}_3 = \mathbf{\Phi}_{-3}^{-1}([\mathbf{x}_0])$ where the initial condition is $[\mathbf{x}_0] = [0, 1] \times [2, 3]$.

Preliminary work

Following the steps described above we have:

Step 1: We will take as initial condition for the reference $\mathbf{a}_0 = (0, 1)^\intercal$. Hence, we get as analytical expression for $\mathbf{a}(t)$,

$$\mathbf{a}(t) = \begin{pmatrix} t \\ e^{-t} \end{pmatrix} \tag{5.7}$$

Step 2: Concerning the second step, we found in Section 4.4.3, that a transport function for Equation (5.6) was

$$\mathbf{h}(\mathbf{x}, \mathbf{a}) = \begin{pmatrix} x_1 - a_1 \\ \frac{x_2}{a_2} \end{pmatrix} \tag{5.8}$$

Step 3: Now, using Proposition 5.2 and the transport function we obtain the following flow function:

$$\begin{aligned} \mathbf{\Phi}_t(\mathbf{x}) &= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a}_0)} \circ \mathbf{a}(t) \\ &= \mathbf{g}_{x_1,x_2} \circ \begin{pmatrix} t \\ e^{-t} \end{pmatrix} \\ &= \begin{pmatrix} t + x_1 \\ x_2 \cdot e^{-t} \end{pmatrix}. \end{aligned}$$

To obtain an inner and an outer approximation of $\mathbb{X}_3$ we will use a separator in the SIVIA algorithm given below. It takes as input the search space to explore $\mathbb{M}$,

the separator associated with $\mathbf{\Phi}_{-3}$, $\mathcal{S}_{\mathbf{\Phi}_{-3}}$, and the precision we want for the SIVIA ϵ. The separator $\mathcal{S}_{\mathbf{\Phi}_{-3}}$ will aim at differentiating the two sets $\mathbb{S}, \bar{\mathbb{S}}$ such that:

$$\mathbb{S} = \left\{ \begin{array}{c} \mathbf{y} = \mathbf{\Phi}_{-3}(\mathbf{x}), \mathbf{x} \in \mathbb{M} \\ \mathbf{y} \in [\mathbf{x}_0] \end{array} \right\} = \mathbb{X}_3 \text{ and } \bar{\mathbb{S}} = \left\{ \begin{array}{c} \mathbf{y} = \mathbf{\Phi}_{-3}(\mathbf{x}), \mathbf{x} \in \mathbb{M} \\ \mathbf{y} \notin [\mathbf{x}_0] \end{array} \right\}. \quad (5.9)$$

Indeed, we are using $\mathbf{\Phi}_{-3}$ as the only thing we are sure of is the initial condition $[\mathbf{x}_0]$. Thus a vector $\mathbf{x}$ will belong to $\mathbb{X}_3$ only if its image when going 3 units back in time belongs to $[\mathbf{x}_0]$.

Algorithm 2 SIVIA($\mathbb{M}$,$\mathcal{S}_{\mathbf{\Phi}_{-t}}$,$\epsilon$)

$s \leftarrow \emptyset$ ▷ Initialise an empty stack e to store boxes to be treated by SIVIA
$s \leftarrow \mathbb{M}$ ▷ Add the search space to the stack
$\mathbb{K}_{in}, \mathbb{K}_{out}, \delta\mathbb{K} = \emptyset$ ▷ Three empty subpavings to store inner boxes, outer boxes and remainder
while s is not empty **do**
  $[\mathbf{x}] \leftarrow$ pop(s) ▷ Remove the last element from the stack s and store it in $[\mathbf{x}]$
  $[\mathbf{x}_{in}], [\mathbf{x}_{out}] = \mathcal{S}_{\mathbf{\Phi}_{-t}}([\mathbf{x}])$ ▷ Apply the separator on $[\mathbf{x}]$
  if $[\mathbf{x}_{in}]$ is empty **then**
    $\mathbb{K}_{in} \leftarrow \mathbb{K}_{in} \cup [\mathbf{x}]$ ▷ $[\mathbf{x}]$ is added to the list of inner boxes
  else if $[\mathbf{x}_{out}]$ is empty **then**
    $\mathbb{K}_{out} \leftarrow \mathbb{K}_{out} \cup [\mathbf{x}]$ ▷ $[\mathbf{x}]$ is added to the list of outer boxes
  else
    if width($[\mathbf{x}]$)$\geq \epsilon$ **then** ▷ Compare the largest dimension of $[\mathbf{x}]$ to ϵ
      $p_1, p_2 =$ bisect($[\mathbf{x}]$) ▷ bisect $[\mathbf{x}]$ along its largest dimension and store the two boxes
      $s \leftarrow p_1$
      $s \leftarrow p_2$
    else
      $\delta\mathbb{K} \leftarrow \delta\mathbb{K} \cup [\mathbf{x}]$ ▷ $[\mathbf{x}]$ is added to the list of remainder boxes
    end if
  end if
end while
return $\mathbb{K}_{in}, \mathbb{K}_{out}, \delta\mathbb{K}$

Implementation

The code used to compute the result illustrated in Figure 5.2 is given in Listing 5.1. It takes about 329 ms and 658 bisections to compute the three different subpavings (outer, inner and remainder) with an accuracy of 0.01. But the real advantage of such a method is that both inner and outer approximation are available with the use of separators when the conventional tools only give an enclosure *i.e* an outer approximation of the result.

Listing 5.1 Characterising $\mathbb{X}_3$ using the Lie integration method

```
// The uncertain initial condition
IntervalVector x_0({{0,1},{2,3}});

// The space to explore for the set inversion
IntervalVector m({{-0.1,6.5},{-0.2,3.5}});

double epsilon = 0.01; // define accuracy of paving

// define transformation function
Function phi("x1","x2","t","(x1+t;x2*exp(-t))");

// Create the general separator on phi_t with [x_0] as constraint
SepFwdBwd SepPhi(phi,x_0);

// Define the time for which we want to perform the integration
Interval t(-3,-3);

// Create the projected separator object
SepProj sepProj(SepPhi,t,epsilon);

// Perform the set inversion algorithm
vector<vector<IntervalVector>> pavings = sivia(m,sepProj,epsilon);
```

As one can see in Listing 5.1, as we already have an analytical expression for $\mathbf{a}(t)$, we do not need to use a guaranteed integration tool to compute an enclosure of the reference. Everything is inside the expression of $\mathbf{\Phi}_{-t}(\mathbf{x})$ but it will be of use later in this work.

Results

The result of the integration is presented in Figure 5.2. In this figure and all the other depicting results of SIVIA algorithms the inner approximation is painted in white, the outer approximation is painted blue when the remainder is in pink. In addition the reference trajectory is painted black evolving towards green. The initial condition is painted in light green. One should notice that the initial condition $\mathbf{a}_0$ of the reference does not need to be contained in the initial box we want to integrate for the method to work.

5.2.2 Integration of multiple discrete sets

In the previous section we presented the method applied on the case of the search of one finite set. However, in most cases, especially in robotics, one wants to know the evolution through time of the system considered. This may be done in a discrete or continuous way. This can be easily done with the new method. We mentioned in Section 2.3.6, that it was possible to combine separators, using union or intersection. We have seen in the previous section that to characterise a set $\mathbb{X}_t$ we were using a separator $\mathcal{S}_{\mathbf{\Phi}_{-t}}$. Hence, if we want to

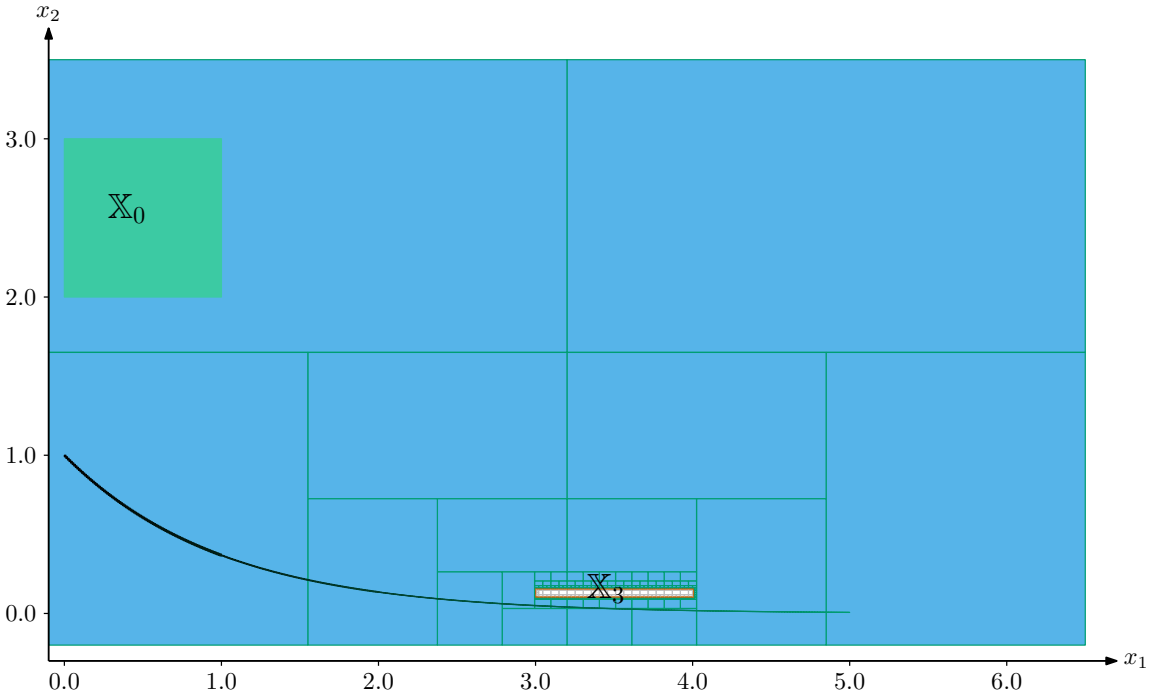


Figure 5.2: Guaranteed integration method applied to TC1.

characterise different sets $\mathbb{X}_{t_i}, i \in \{1, \ldots, m\}$ we can do so by creating another separator

$$\mathcal{S} = \bigcup_{i \in \{1 \ldots m\}} \mathcal{S}_{\mathbf{\Phi}_{-t_i}}, \tag{5.10}$$

where each $\mathcal{S}_{\mathbf{\Phi}_{-t_i}}$ is the separator used to approximate $\mathbb{X}_{t_i}$.

Example 5.2 (Integration TC2: discrete case)**.** Consider the system defined in TC2 seen in Example 4.11:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} 1 \\ \sin(x_1) \end{pmatrix}. \tag{5.11}$$

Preliminary work

Once again, let us perform the three different steps of our integration method.

Step 1: We take as initial condition for the reference $\mathbf{a}_0 = (0, 0)^\intercal$. From this, we have as analytical expression for $\mathbf{a}(t)$:

$$\mathbf{a}(t) = \begin{pmatrix} t \\ 1 - \cos t \end{pmatrix}. \tag{5.12}$$

Step 2: We found a valid transport function for Equation (5.11) in Example 4.11. We recall it here:

$$\mathbf{h}(\mathbf{x}, \mathbf{a}) \begin{pmatrix} x_2 - \mathbf{\Phi}_{x_1 - a_1, 2}(\mathbf{a}) \\ x_1 - a_1 \end{pmatrix}. \tag{5.13}$$

Step 3: Lastly, from the two first steps, we can determine the flow function $\mathbf{\Phi}_t(\mathbf{x})$. On the one hand, we have $\mathbf{\Phi}_t(\mathbf{0}) = \mathbf{a}(t)$ (no displacement from the reference). Therefore $\mathbf{\Phi}_{x_1}(\mathbf{0}) = \mathbf{a}(x_1)$. Thus

$$\mathbf{h}(\mathbf{x}, \mathbf{0}) \overset{(5.13)}{=} \begin{pmatrix} x_2 - \mathbf{\Phi}_{x_1,2}(\mathbf{0}) \\ x_1 \end{pmatrix} = \begin{pmatrix} x_2 - a_2(x_1) \\ x_1 \end{pmatrix}. \tag{5.14}$$

On the other hand:

$$\mathbf{g_p}(\mathbf{x}) = \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \mathbf{\Phi}_{p_2}(\mathbf{x}) \tag{5.15}$$

Hence,

$$\begin{aligned} \mathbf{g_p} \circ \mathbf{a}(t) &= \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \mathbf{\Phi}_{p_2}(\mathbf{a}(t)) \\ &= \begin{pmatrix} 0 \\ p_1 \end{pmatrix} + \mathbf{a}(t + p_2) \\ &= \begin{pmatrix} a_1(t + p_2) \\ p_1 + a_2(t + p_2) \end{pmatrix}. \end{aligned}$$

Combining both part we obtain:

$$\mathbf{\Phi}_t(\mathbf{x}) = \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{0})} \circ \mathbf{a}(t) \tag{5.16}$$

$$= \begin{pmatrix} a_1(t + x_1) \\ x_2 - a_2(x_1) + a_2(t + x_1) \end{pmatrix} \tag{5.17}$$

$$= \begin{pmatrix} t + x_1 \\ x_2 + \cos x_1 - \cos(t + x_1) \end{pmatrix}. \tag{5.18}$$

With this analytical formulation of $\mathbf{\Phi}_t(\mathbf{x})$, we can easily build one separator $\mathcal{S}_{t_i}$ for each set $\mathbb{X}_{t_i}$ we want to characterise as done in Example 5.1. Then the final separator is simply the union of all the $\mathcal{S}_{t_i}$s. The result of the integration for an initial condition $[\mathbf{x}_0] = [0, 1]^2$ is given in Figure 5.3.

Implementation

We will not display the full code of the example here as it is very similar to what have been presented in Listing 5.1. One need to change the value of the initial condition, the search space explored and the expression of $\mathbf{\Phi}_t$. Here we will only focus on creating the final separator $\mathcal{S}$ which is the union of the different $\mathcal{S}_{t_i}$. The lines presented in Listing 5.2 should replace the ones presented in Listing 5.1 starting at line 17.

Results

The computed solution sets are depicted in Figure 5.3. One might have a hard time to figure out the continuity between the different sets computed, that is why in Figure 5.4 we added the full trajectories of the corners of the initial box. A video of the full set displacement is available here[1].

[1] Youtube: Visualising set flow with Lie Groups by Julien DAMERS

Listing 5.2 Performing integration for several discrete sets

```
    /*
    * Define the initial condition
    * Define the search space m
    * Give the desired time step
    * Indicate the analytic form of phi_t
    * Create the general separator S associated with phi_t with the constraint on x_0
    */
    // Define the different times on which we want to integrate
    vector<Interval*> t_s{
        new Interval(0.),
        new Interval(-2.),
        new Interval(-4.),
        new Interval(-6.),
        new Interval(-8.)};

    vector<Sep*> seps;

    // Generate the separator for each individual time and store them in seps
    for (size_t i=0;i<projections.size();i++)
    {
        SepProj *sepProj = new SepProj(*SepPhi,*(t_s[i]),epsilon);
        seps.push_back(sepProj);
    }

    //Create the union of all the seprators
    Array<Sep> ar_sep(seps);
    SepUnion usep (ar_sep);


    // Perform the set inversion algorithm
    vector<vector<IntervalVector>> pavings = sivia(m, usep, epsilon);
```

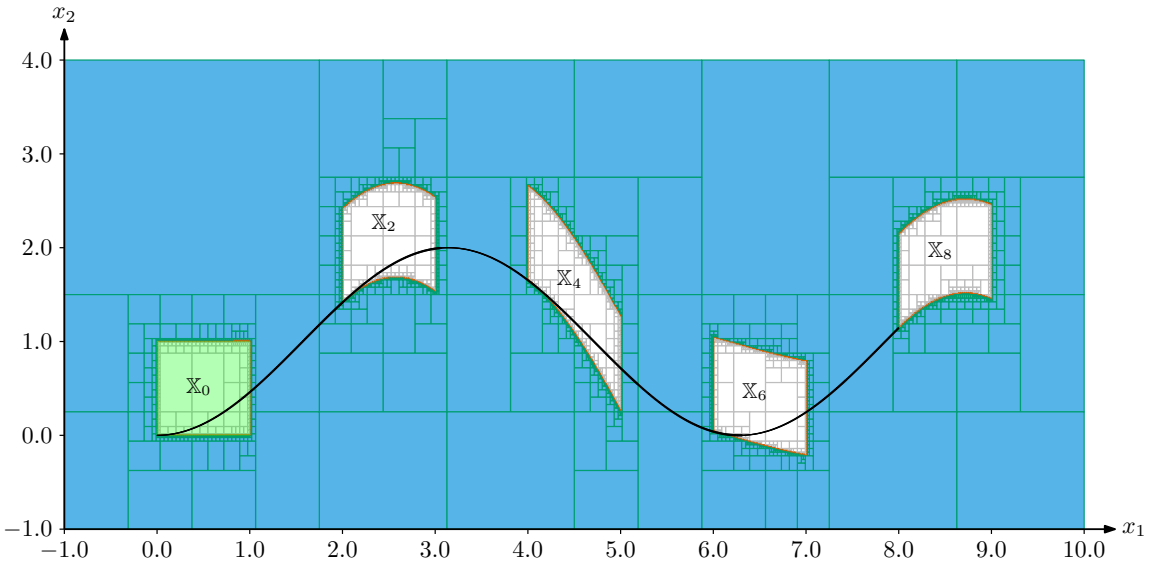


Figure 5.3: Result of integration for TC2 for a discrete time set. Here is computed the different $\mathbb{X}_{t_i}$ for $t_i \in \{0, 2, 4, 6, 8\}$.

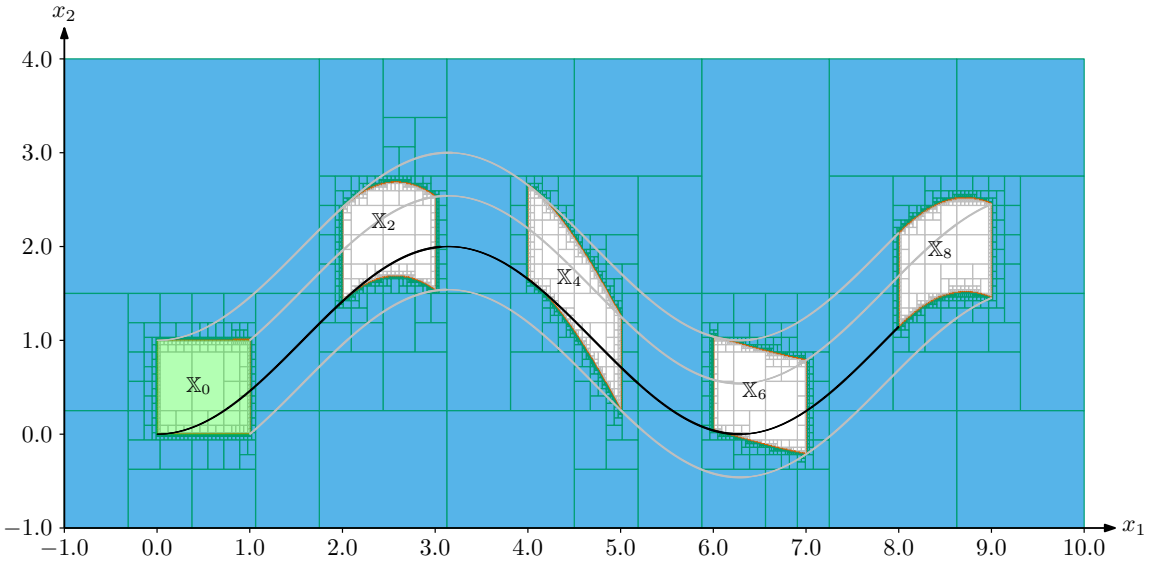


Figure 5.4: Explanation of the set flow: We represented the full trajectories of the the four corners of the initial box. One should notice that the solution sets are always contained in the upper and lower trajectory boundaries

5.2.3 Continuous integration

In the previous sections, we focused on computing discrete sets. However, if this is most of the time less computationally heavy, it might not be enough, and one might want to compute an inner and outer approximation for the complete trajectory. Thus instead of computing:

$$\mathbb{X} = \bigcup_{t_i \in \{t_1 \dots t_m\}} \mathbb{X}_{t_i}, \tag{5.19}$$

one needs to compute:

$$\mathbb{X} = \bigcup_{t \in [t_0, t_{max}]} \mathbb{X}_t, \tag{5.20}$$

This can be done with our new method using a particular tool called the projection of separators presented in the next section.

5.2.3.1 Projection of sets

The system considered in TC2 (or TC1), when presented under the state equation form is a bi-dimensional problem. However, when integrating the system, a new variable comes into place, the time t. In both previous examples, t was fixed as we were considering a discrete case, hence it could be considered as a fixed parameter and not a variable, our problem was still in two dimensions. Nevertheless the set $\mathbb{X}$ we want to characterise in Equation (5.20) is bi-dimensional thus we will need to project our new 3D problem into a 2D one. We introduce the set $\mathbb{S}$ such that:

$$\mathbb{S} = \{(\mathbf{x}, t), \mathbf{x} \in \mathbb{R}^2, t \in [t_0, t_f] | \mathbf{\Phi}_{-t}(\mathbf{x}) \in [\mathbf{x}_0]\}. \tag{5.21}$$

Obviously, $\mathbb{S}$ gathers all the points of trajectories satisfying both the state equation and the initial condition. The projection of $\mathbb{S}$ on $\mathbb{R}^2$ is

$$\mathbb{S}_{\mathbb{R}^2} = \{\mathbf{x} \in \mathbb{R}^2 | \exists t \in [t_0, t_f], \mathbf{\Phi}_{-t}(\mathbf{x}) \in [\mathbf{x}_0]\} = \mathbb{X}. \tag{5.22}$$

Therefore, we need a tool to compute an approximation of our desired set $\mathbb{X}$ and solve our problem. This tool is the projection of separators developed in IBEX inspired by [64]. The projection algorithm can be seen as a SIVIA algorithm where the element of $\mathbb{R}^2$ is fixed and the variable t is the one dimension along which the SIVIA is performed.

5.2.3.2 Continuous integration: an example

Let us come back to TC2 that we left in Section 5.2.2. We will now perform the integration in a continuous manner.

Example 5.3 (Integration TC2: continuous case)**.** Once again consider the system defined in TC2.

Preliminary work

Obviously, the preliminary work is exactly the same as the one done in Example 5.2. Hence the analytic form of the flow function is the same, thus the general separator $\mathcal{S}_{\mathbf{\Phi}_{-t}}$ stays unchanged. What is different now is the implementation.

Implementation

The careful reader will have noticed that in Listing 5.1 and Listing 5.2, we were already using the projection of separator but not at its full potential only for ease of use. Indeed we were using degenerate intervals (*i.e* a scalar values) to project our 3D problem with x_1, x_2, t to a 2D map using x_1, x_2 coordinate system. Here we will use an interval as input for our projector object. The lines 12 to 31 of Listing 5.2 will be substituted by the ones presented Listing 5.3.

Listing 5.3 Performing integration on a continuous time interval (Codac V1)

```
#import "codac.h"

int main()
{
    /*
    * Define the initial condition
    * Define the search space m
    * Give the desired time step epsilon
    * Indicate the analytic form of phi_t
    * Create the general separator S associated with phi_t with the constraint on x_0
    */

    // Define the time interval on which we want to integrate
    Interval t(-8,0);

    // Create the separator object
    SepProj sepProj(SepPhi,t,epsilon);

    // Perform the set inversion algorithm
    vector<vector<IntervalVector>> pavings = sivia(m,sepProj,epsilon);

    // Graphics ...
}
```

Results

The result of the continuous integration with an initial condition $[\mathbf{x}_0] = [0,1]^2$ over a time interval $[0,8]$ is depicted in Figure 5.5.

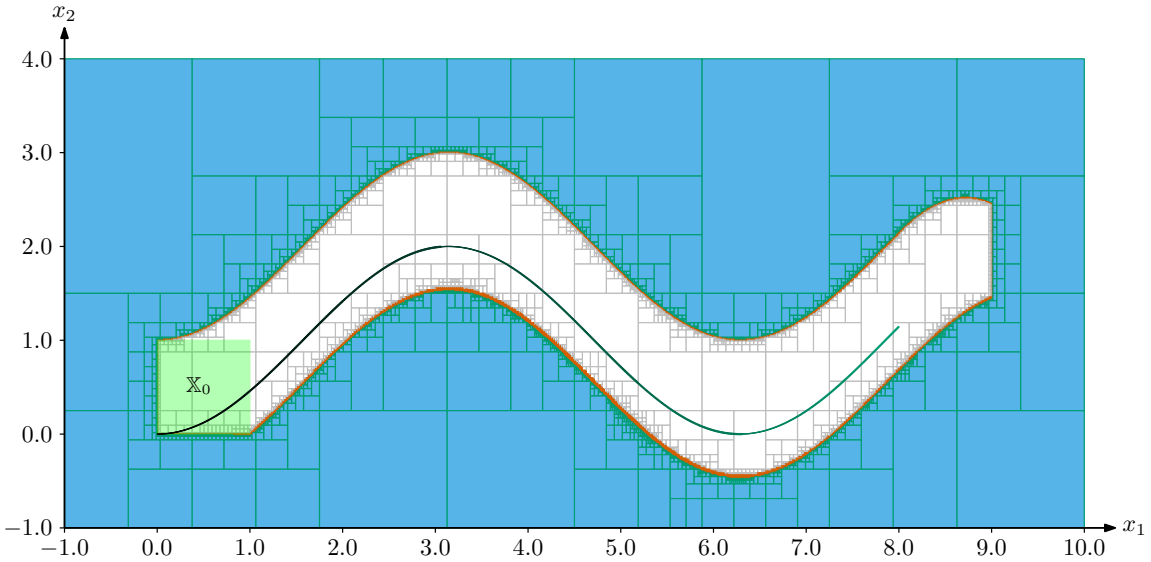


Figure 5.5: Continuous integration in TC2

5.3 Contraction of tubes

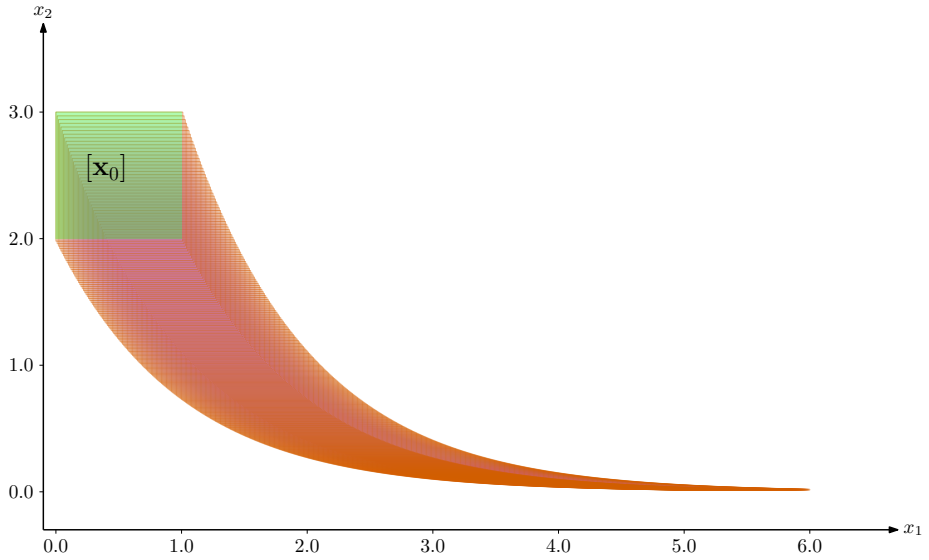
In the previous sections, we applied the SIVIA algorithm on a search space in both discrete and continuous case. When solving the latter, and considering the union of the remainder $\partial\mathbb{X}$ and the inner approximation $\mathbb{X}^-$, we basically compute an outer enclosure of all possible trajectories that satisfy the state equation and start from a point contained in the initial box. These trajectories could be enclosed in a tube instead of their projection in a union of subpavings. Thus, extending the work presented in the Section 5.2, we apply our integration method to tubes. This time, only an outer approximation will be computed as separators cannot be applied in the scope of tube theory.

Example 5.4 (Integration TC1: Tube)**.** In this example, we will consider the system defined in TC1. Again, all the preliminary work made in Example 5.1 still holds, especially the analytical form of $\mathbf{\Phi}_t$. What differs is, instead of generating a separator from $\mathbf{\Phi}_t$, we will only create the contractor $\mathcal{C}_{\mathbf{\Phi}_{-t}}$ on the inner set *i.e* the contractor that removes non solutions. As a tube is composed of slices, which are boxes, we will apply the contractor on each slice, using as parameter t of $\mathbf{\Phi}_t$, the interval $[k * dt, (k+1) * dt]$ where k is the index of the slice, and dt the width of the time domain for each slice. This is illustrated in Listing 5.4.

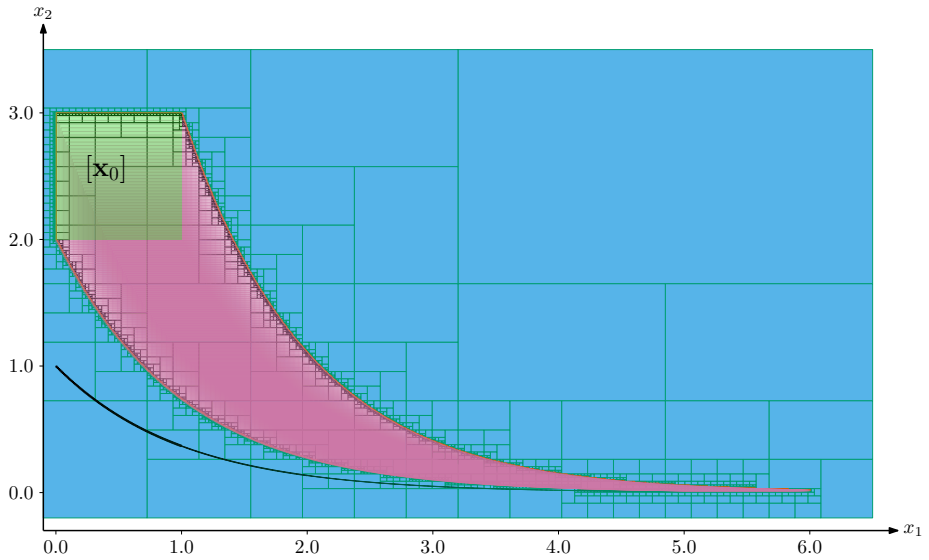
One may notice that the expression for $\mathbf{\Phi}_t$ is slightly different in Listing 5.4 compared to the one used in Listing 5.1. Indeed, in Listing 5.1, it was possible to indicate negative values for the time. In this format, we use the time encapsulated in the tube object which is always positive. Thus, t is replaced by $-t$ in the expression of $\mathbf{\Phi}_t$. The result obtained with this piece of code is presented in Figure 5.6. We added the continuous integration using the method presented in Section 5.2.3 as a background to prove the efficiency.

Listing 5.4 Performing integration with tubes (Codac V1)

```
#import "codac.h"

int main()
{
    // Define time domain of integration and the time step
    Interval time_domain(0,5);
    double timestep = 0.01;

    // Create a generic tube
    TubeVector x_lie(time_domain, timestep, 2);

    // Indicate the initial condition and the phi_t function
    IntervalVector x_0({{0,1},{2,3}});
    Function phi_t("t","x1","x2","(x1-t;x2*exp(t))");

    // Create the contractor associated with phi_t on a box with initial condition x_0
    CtcFwdBwd c_phi(phi_t,x_0);

    /*
     * Make the contractor applicable to a tube (tell it to apply ctc_phi on each
     * slice of the tube)
     */
    CtcStatic c_phi_tube(c_phi, true);

    // Contract
    c_phi_tube.contract(x_lie);

    // Graphics ...
}
```



(a) Output of Listing 5.4



(b) Comparison of the tube computed with the use of the SIVIA algorithm

Figure 5.6: Integration method applied on a tube: Figure 5.6a depicts the output of the code presented in Listing 5.4. Figure 5.6b, shows the same tube as Figure 5.6a with a background computed using the SIVIA algorithm to approximate $\bigcup_{t\in[0,5]} \mathbb{X}_t$. One should notice that the computed tube is equal to the outer approximation of the solution step *i.e*, the union of the inner approximation and the remainder.

5.4 Comparisons with Löhner contractor, CAPD, Flow*

We compare this integration method to the already existing ones in order to assess its efficiency. There are several key aspects that we will analyse here: computer processing time and robustness regarding the size of the initial condition. In this section, we will compare our method to CAPD and the Löhner contractor integrated in Codac already presented in Chapter 3 and Flow* [20], a reachability analysis tool using guaranteed integration. We would like to thank Dr. Chen who kindly helped us with the use of Flow*. Also, we will compare our method when contracting tubes (denoted Lie (tubes)) to the others. But we remind that our method is able to return both an inner and an outer approximation when the others only provide an outer approximation.

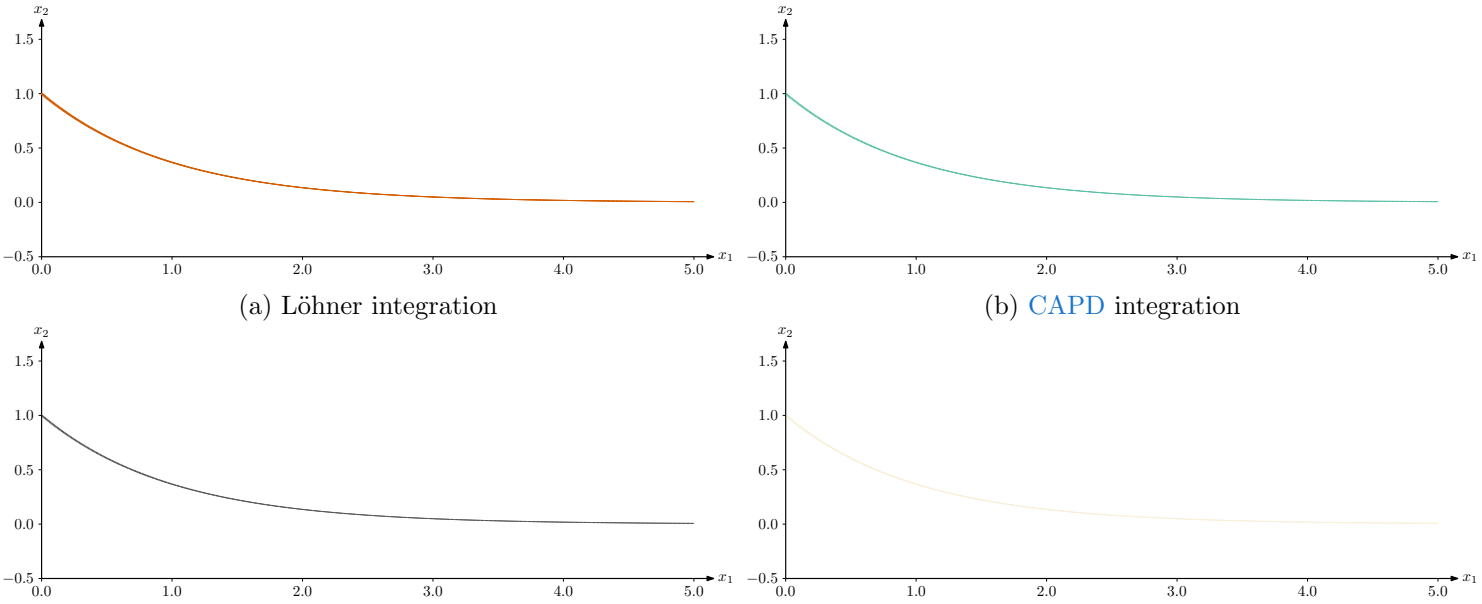
5.4.1 Comparing computer processing time

First let us consider the system of TC1. We will use as initial condition for the integration the singleton $\mathbf{a}_0$ used for the reference. We will integrate over the time interval $T = [0, 5]$ with a time step $dt = 0.1$. The results obtained with each tool are presented in Table 5.1.

Experiment: TC1, $[\mathbf{x}_0] = \{\mathbf{a}_0 = (0, 1)\}$, $T = [0, 5]$, $dt = 0.01$, Taylor expansion order= 5	
Tool	**computer processing time** (average)
Löhner	25 ms
CAPD	3.7 ms
Flow*	164 ms
Lie (tube)	4.4 ms (3.7 ms reference computation + Lie integration 0.7 ms)

Table 5.1: Experiment parameters to measure computer processing time on TC1

As it can be seen in Figure 5.7, all methods compute the same results with the same accuracy. As expected, out of the conventional integration tools, CAPD obtains the best results by far in terms of processing time. This was not surprising as it is considered as the state of the art in the field. However, our method manages to compute the same result in almost five times faster, provided we have a reference. This shows that not having to process step by step is a real advantage. However, in this small case the full processing time should take into account that we need to compute the reference before applying our method. Thus CAPD performs the best when considering a single point as initial condition. It is even possible to increase the speed of CAPD (and thus the one of Lie integration) using one particular feature of CAPD which is the adaptive time step. This allows the solver to choose the time step for each step in order to increase the speed of the algorithm (without loosing in enclosure sharpness). In this particular experiment the processing time is almost divided by two, with an average processing time of 1.9 ms.



(a) Löhner integration

(b) CAPD integration

(c) Flow* integration

(d) Lie integration method

Figure 5.7: Comparison of outputs when integrating a single point in TC1

We also did the same experiment using the system defined in TC2. The results are presented in Table 5.2 below. Again the results computed are the same for all methods in terms of sharpness but the processing times differ.

Experiment: TC2, $[\mathbf{x}_0] = \{\mathbf{a}_0\}$, $T = [0, 15]$, $dt = 0.01$, Taylor expansion order= 5	
Tool	**computer processing time** (average)
Löhner	173 ms
CAPD	21 ms
Flow*	700 ms
Lie (tube)	27 ms (reference computation 21 ms + Lie integration 6 ms)

Table 5.2: Experiment parameters to measure computer processing time on TC2

5.4.2 Robustness

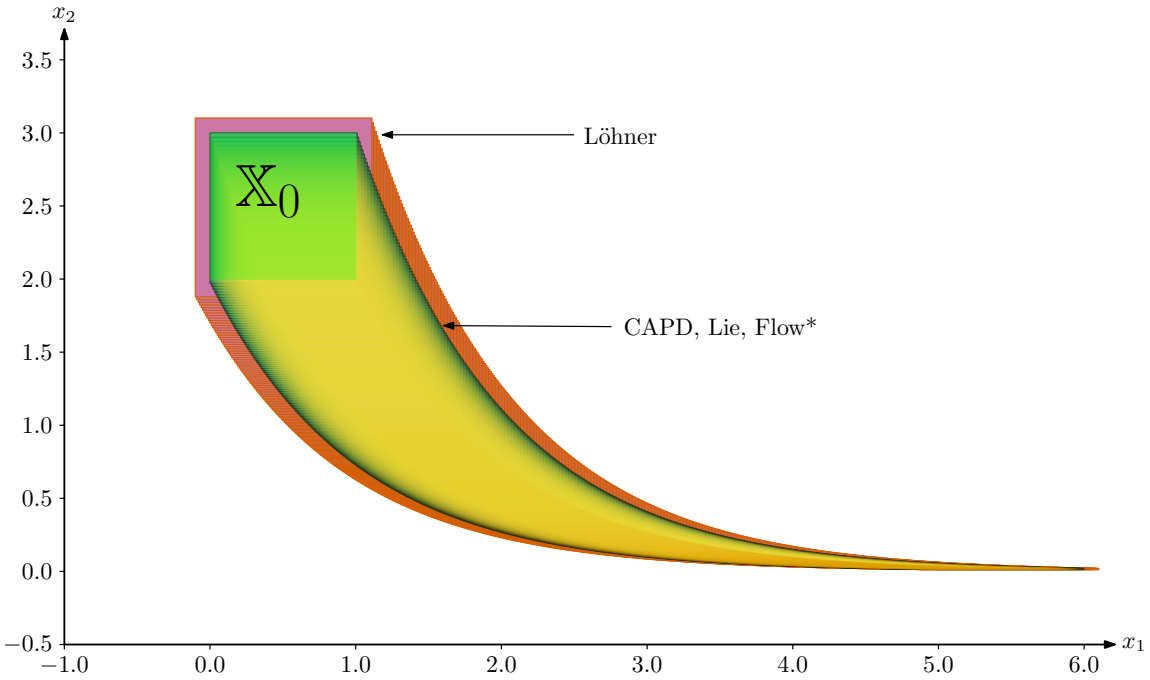
This second comparison will be a major one, more than computer processing time, as it is one that motivated this research. By robustness, we mean the ability to compute a sharp enclosure of the trajectory when the uncertainty on the initial condition increases. This will be done by measuring the evolution of the volume of the box at each step of the computation. When dealing with intervals, the volume of the box is the multiplication of the width of the interval on each of its dimension. The computer processing time is a second criterion. If a method is not robust enough, then the bloating effect seen in Section 3.4.5 might appear. As mentioned in Chapter 2, in robotics, the initial condition of the system is, in general, uncertain. Thus having a guaranteed integration tool that is robust to the

increase of the size of the initial condition is of great interest. Once again we will use TC1 and TC2. The parameters of both experiments are given in Table 5.3 and Table 5.4.

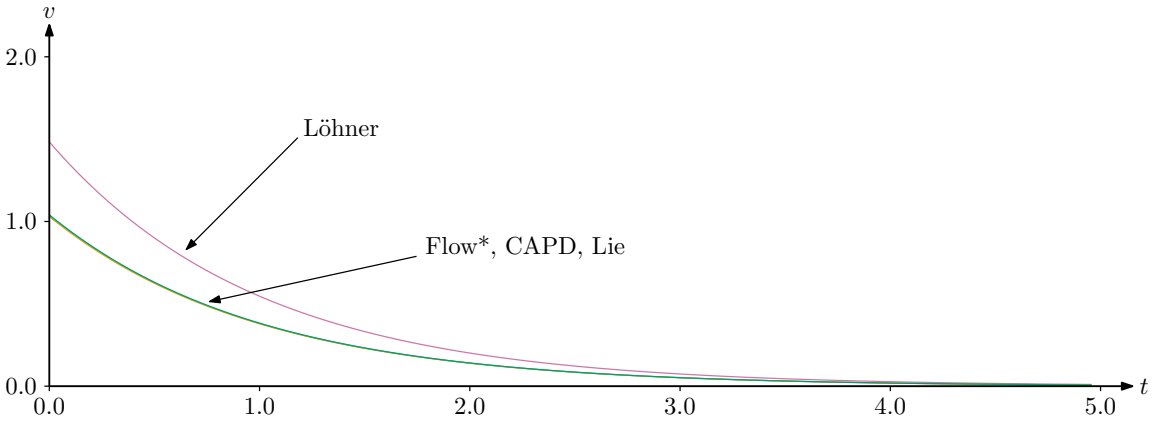
Experiment: system from TC1, $[\mathbf{x}_0] = [0,1] \times [2,3]$, $T = [0,5]$, $dt = 0.01$	
Tool	**computer processing time (average)**
Löhner	173 ms
CAPD	3.7 ms
Flow*	200 ms
Lie (tube)	4.4 ms (reference computation 3.7 ms + Lie integration 0.7 ms)
Lie (SIVIA)	140 ms

Table 5.3: Robustness measurements on TC1

There is a slight increase in the processing time for all methods. We also indicated the processing time when applying the SIVIA algorithm, thus obtaining an inner approximation. One can notice that, if we cannot beat CAPD we are still getting better results that Flow* and the Löhner contractor while having an inner and outer approximation which the other two cannot provide. Figure 5.8 shows the results of the different tools. One can directly notice that the Löhner contractor is the one that is the less robust when all other methods returns almost the same enclosure. The Lie integration method gives the sharpest one followed by CAPD and then Flow* Overall, in this simple case, all methods are quite robust against the increase of the volume of the initial condition.



(a) Enclosure of the trajectory of system TC1 using different methods (Löhner (red), CAPD (blue), Flow* (green), Lie (yellow))



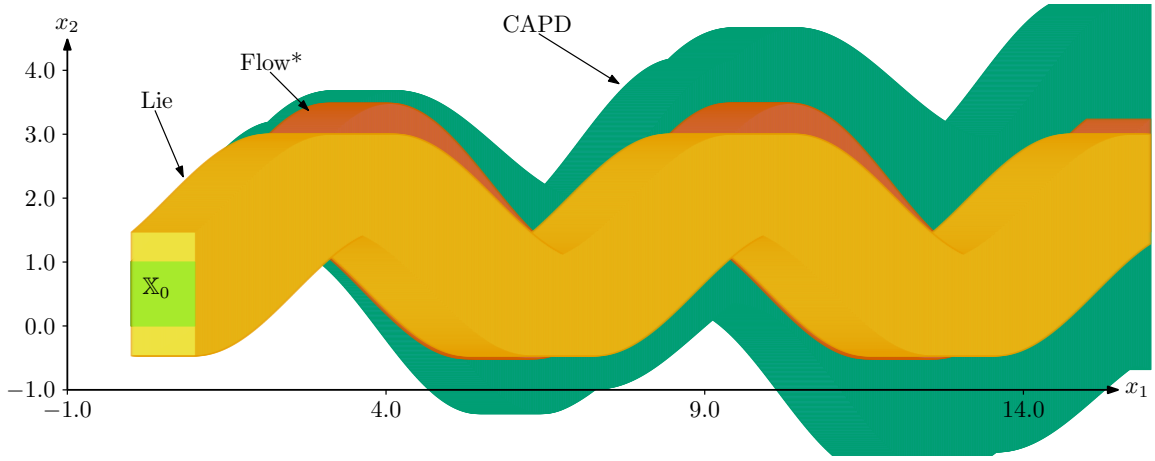
(b) Evolution of the volume of the enclosures computed with different methods

Figure 5.8: Robustness comparison on TC1

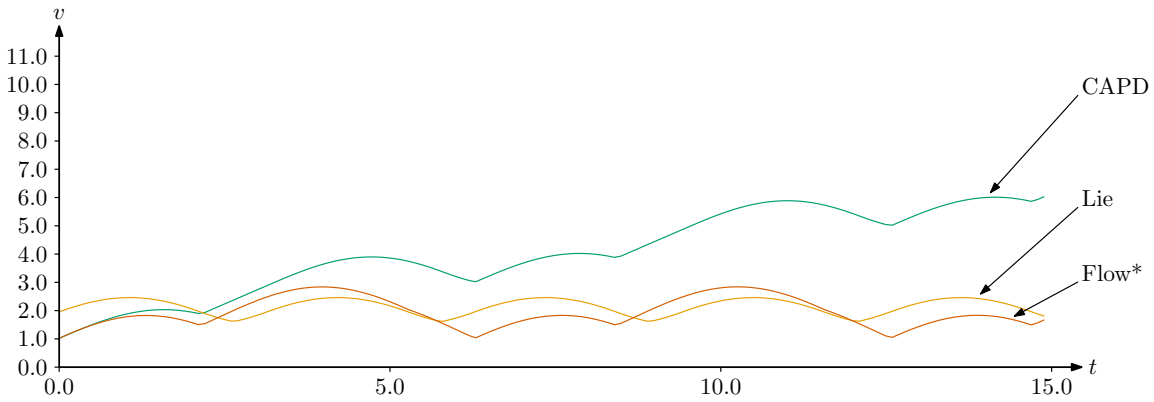
We then performed the same kind of experiment but on the second test case. This time, the simple Löhner contractor could not return an enclosure of the trajectory, over the 15 s of integration. The different tubes computed by the other methods are presented in Figure 5.9 and the associated processing times are displayed in Table 5.4. The bloating effect starts appearing with CAPD. Flow* and the Lie integration methods are far more robust in this case. This is expected as Flow* is designed to perform reachability analysis. In this field, large initial conditions are more frequent hence the tool is implemented such that it can handle this problem. In addition, the reader probably noticed that the enclosure of Flow* is sharper than the one returned by the Lie integration method at the beginning of the tube. This is due to the interval manipulation which adds pessimism. Performing a SIVIA solve this problem as we have already seen in Figure 5.5.

Experiment: system from TC2, $[\mathbf{x}_0] = [0,1] \times [0,1]$, $T = [0,15]$, $dt = 0.01$	
Tool	**computer processing time**
Löhner	Cannot compute, too much bloating effect
CAPD	27 ms
Flow*	1.2 s
Lie (tube)	26 ms (reference computation 21 ms + Lie integration 5 ms)
Lie (SIVIA)	7.8 s

Table 5.4: Robustness measurements on TC2



(a) Enclosure of the trajectory of system TC2 using different methods (CAPD (green), Flow* (orange), Lie (yellow)). One can notice that the Lie integration method does not fit exactly the initial condition depicted in green. This is due to multiple occurrences in the flow function (Equation (5.16))



(b) Evolution of the volume of the enclosures computed with different methods

Figure 5.9: Robustness comparison on TC2

5.4.3 Scaling

One other aspects on which the different methods can be compared is the ability to perform guaranteed integration repeatedly that we will denote by scaling. Indeed when working with simulations, we may have to compute the evolution of a system many times. Thus the ability to do it within a minimum time becomes interesting. We compared the performances of the different methods on both TC1 and TC2 with a thousand particles. Each particle was contained in the box used for the robustness test done previously. The results are summed up in Table 5.5 and Table 5.6. One can notice that this is one main advantage of using Lie groups for guaranteed integration. As it does not need to integrate step by step but only transforms a reference, computing the solution requires less operations thus a lower processing time. Hence the Lie integration method suits well when working on a greater scale. One can envision to use the Lie integration method in other area rather than with

interval analysis. We believe this can be a great interest for particle filters where numerous integrations need to be performed to estimate localise a robot. It could probably help in the case of real time applications.

Experiment	
Tool	**Computer processing time** (average)
Löhner	25 s
CAPD	3.7 s
Flow*	164 s
Lie (tube)	703 ms

Table 5.5: TC1 scaling ability comparison

Experiment	
Tool	**Computer processing time** (average)
Löhner	173 s
CAPD	21 s
Flow*	164 s
Lie (tube)	6 s

Table 5.6: TC2 scaling ability comparison

5.4.4 Computing an inner approximation

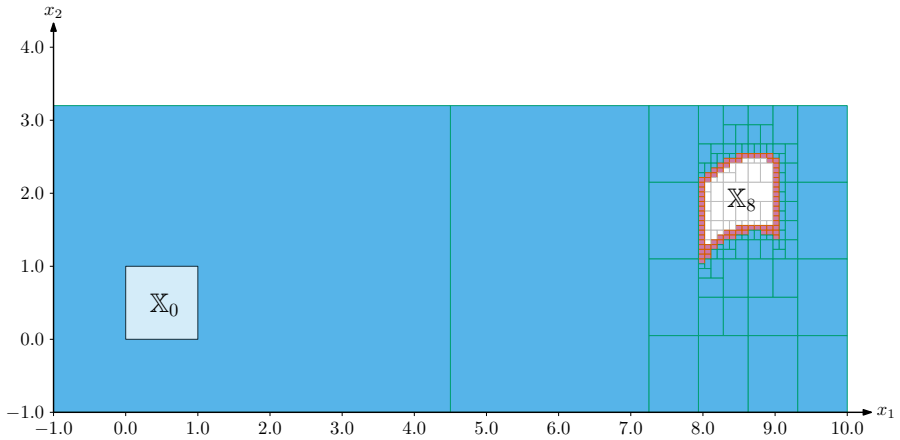
In this last section we will compare the ability to provide an inner approximation of the result set. If CAPD, Flow* and the Löhner contractor are not designed to do so, it is possible to do it using a *bisect and integrate* method. It works like the SIVIA algorithm. The difference is in the action performed when treating a box from the stack. Instead of applying the separator as done in Algorithm 2 line 6, we will perform the guaranteed integration backward and check if the result belongs to the initial box $[\mathbf{x}_0]$. If it does, then the box is put in the list of inner boxes, if the intersection with the initial box is empty, it goes into the list of outer boxes. It is bisected (or put in the boundary list) otherwise. We used CAPD to compute the approximation of an inner set using the bisect and integrate method. Our metrics for this last comparison will be the computer processing time and the number of bisections.

Experiment: system from TC2, $[\mathbf{x}_0] = [0,1] \times [0,1]$, $T = 8$		
Tool	**comp. time** (average)	**bisections**
CAPD	1364 ms	193
Lie (SIVIA)	8 ms	162

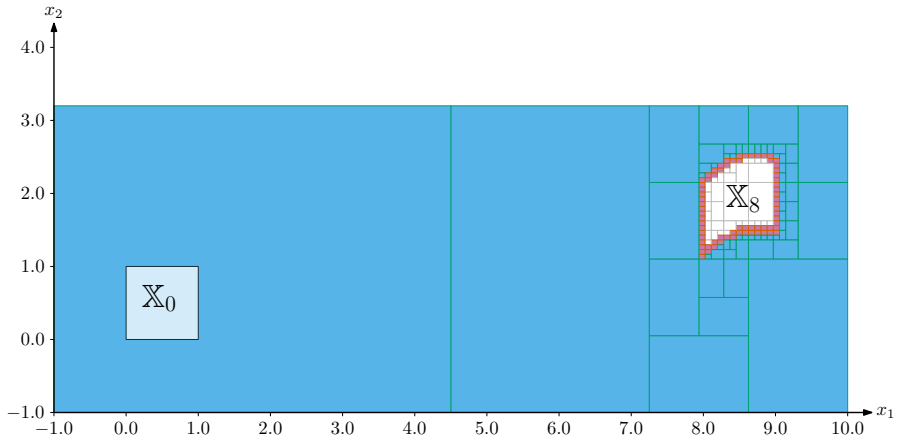
Table 5.7: Comparisons of inner set approximation computation

We have seen in previous sections why the computation time is lower for the Lie integration method. However, one can notice that the number of bisections required is also higher when using CAPD. This can be explained with the result obtained when we were studying

the robustness. As we have seen, CAPD is less robust to the increase of the size of the initial condition. Hence when performing the integration back in time, the chance of the result to be part of the constraint set is lower. Thus CAPD needs to bisect smaller chunks than the Lie integration method, leading to more bisections. The two different outputs are presented in Figure 5.10. One can notice that CAPD needs to bisect in smaller boxes in order to check to which approximation the box belongs to.



(a) Set approximations returned with CAPD.



(b) Set approximations returned with Lie integration method.

Figure 5.10: Comparison of the outer and inner approximations computed between CAPD and the Lie integration method.

5.5 Another example

5.5.1 Introduction to test case 3

Since the beginning of this chapter we have considered two systems, in TC1 and TC2 for which computing an analytical expression of the flow was easy. Moreover finding enough

symmetries, *i.e* independent symmetries for each dimension, was no difficult task. In this section we introduce a new system that we will denote by Test-Case 3 (TC3). This system is defined by

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} -x_1^3 - x_1 * x_2^2 + x_1 - x_2 \\ -x_2^3 - x_1^2 * x_1 + x_1 + x_2 \end{pmatrix}. \tag{5.23}$$

The question of integrating this system is treated in [54, Chapter 1] and [24]. As already shown in Example 4.7, the vector field associated with this system is the one presented in Figure 5.11

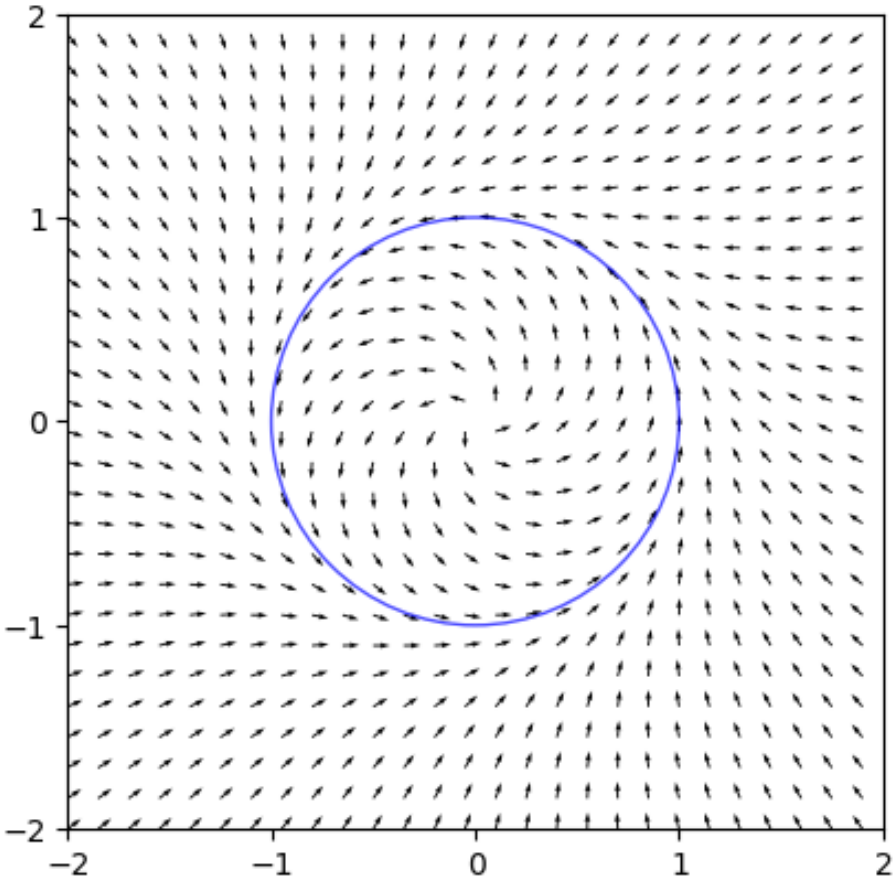


Figure 5.11: Vector field associated with Equation (5.23).

One can notice that all trajectories will converge towards the unit circle painted blue in Figure 5.11. Moreover, we have already figured out that any rotation would be a symmetry for the system. But, as there are two degrees of freedom, we need, at least, another symmetry.

5.5.2 Finding symmetries

In the previous section we have shown that the vector field was converging towards a circle and that one symmetry was the rotation. However this kind of symmetry acts on both axes of our Cartesian coordinate system. Hence, from intuition, one can feel that polar coordinates might suit better to deal with this system. Let us convert our system in these new coordinates. We have:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = r \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$
$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -r\sin\theta & \cos\theta \\ r\cos\theta & \sin\theta \end{pmatrix} \cdot \begin{pmatrix} \dot{\theta} \\ \dot{r} \end{pmatrix}$$

Therefore

$$\begin{aligned} \begin{pmatrix} \dot{\theta} \\ \dot{r} \end{pmatrix} &= \begin{pmatrix} -r\sin\theta & \cos\theta \\ r\cos\theta & \sin\theta \end{pmatrix}^{-1} \cdot \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} \\ &= \begin{pmatrix} -\frac{\sin\theta}{r} & \frac{\cos\theta}{r} \\ \cos\theta & \sin\theta \end{pmatrix} \cdot \begin{pmatrix} -x_1^3 - x_1 * x_2^2 + x_1 - x_2 \\ -x_2^3 - x_1^2 * x_1 + x_1 + x_2 \end{pmatrix} \\ &= \begin{pmatrix} -\frac{\sin\theta}{r} & \frac{\cos\theta}{r} \\ \cos\theta & \sin\theta \end{pmatrix} \cdot \begin{pmatrix} -r^3\cos^3\theta - r^3\cos\theta\sin^2\theta + r\cos\theta - r\sin\theta \\ -r^3\sin^3\theta - r^3\cos^2\theta\sin\theta + r\cos\theta + r\sin\theta \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ -r^3 + r \end{pmatrix} \end{aligned}$$

With the last equality, we observe that the evolution of θ and r are decoupled. Thus we need to find one symmetry for each parameter. As seen in TC1 and TC2 with the parameter x_1, a stabiliser for $\dot{\theta} = 1$ is a translation of the form:

$$g_1(\theta) = \theta + p_1. \tag{5.24}$$

Converted into the Cartesian coordinate system, this corresponds to the rotation of the field.

We now need to find a stabiliser g_2 for the equation $\dot{r} = -r^3 + r$. We know from Section 4.3.2 that for g_2 to be a stabiliser, it needs to satisfy

$$\begin{aligned} \tfrac{dg_2}{dr}(r) \cdot r(1-r^2) &= (r(1-r^2)) \circ g_2(r) \\ &= g_2(r)(1 - g_2^2(r)). \end{aligned} \tag{5.25}$$

Hence we obtain the differential equation:

$$\frac{dg_2}{dr}(r) = \frac{g_2(r)(1-g_2^2(r))}{r(1-r^2)}, \tag{5.26}$$

which is a Bernoulli equation. Using a solver such as Wolfram Alpha we obtain a solution of the form:

$$g_2(r) = \frac{r}{\sqrt{p_2 + r^2(1-p_2)}}, \tag{5.27}$$

where p_2 is a parameter. From g_1 and g_2, we can define $\mathbf{g_p}$ such that

$$\mathbf{g_p}\left(\begin{pmatrix}\theta\\ r\end{pmatrix}\right) = \begin{pmatrix}\theta + p_1\\ \frac{r}{\sqrt{p_2+r^2(1-p_2)}}\end{pmatrix}, \tag{5.28}$$

which is a stabiliser for Equation (5.23) in the polar coordinates. To get its equivalent into the Cartesian coordinate system, we apply Proposition 4.4 with

$$\mathbf{w}\begin{pmatrix}\theta\\ r\end{pmatrix} = r\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix}.$$

Therefore the stabiliser is

$$\begin{aligned}
\mathbf{w}\circ\mathbf{g_p}\circ\mathbf{w}^{-1}\begin{pmatrix}x_1\\ x_2\end{pmatrix} &= \mathbf{w}\circ\mathbf{g_p}\begin{pmatrix}\mathrm{atan2}(x_2,x_1)\\ \sqrt{x_1^2+x_2^2}\end{pmatrix}\\
&= \mathbf{w}\circ\begin{pmatrix}\mathrm{atan2}(x_2,x_1)+p_1\\ \frac{\sqrt{x_1^2+x_2^2}}{\sqrt{p_2+(x_1^2+x_2^2)(1-p_2)}}\end{pmatrix}\\
&= \frac{\sqrt{x_1^2+x_2^2}}{\sqrt{p_2+(x_1^2+x_2^2)(1-p_2)}}\begin{pmatrix}\cos(\mathrm{atan2}(x_2,x_1)+p_1)\\ \sin(\mathrm{atan2}(x_2,x_1)+p_1)\end{pmatrix}\\
&= \frac{1}{\sqrt{p_2+(x_1^2+x_2^2)(1-p_2)}}\cdot\mathbf{R}_{p_1}\cdot\begin{pmatrix}x_1\\ x_2\end{pmatrix}.
\end{aligned} \tag{5.29}$$

In order to keep the same notations, we will also denote by $\mathbf{g_p}$ the stabiliser in the Cartesian coordinate system. The action of $\mathbf{g_p}$ is depicted in the following example.

Example 5.5 (TC3: group action)**.** Let us test the symmetry we found above with a trajectory $\mathbf{a}(\cdot)$. If $\mathbf{a}(\cdot)$ is solution of Equation (5.23) then every $\mathbf{g_p}(\mathbf{a}(\cdot))$ is a solution as $\mathbf{g_p}$ is a symmetry. However one can wonder if $\mathbf{g_p}$ is always defined as we have a square root in its expression. To illustrate this, we will arbitrarily take as initial condition for $\mathbf{a}(\cdot)$, $\mathbf{a}(0) = (\frac{1}{2}, 0)$. Then, we will compute solutions for various values of $\mathbf{p}$:

$$\mathbf{p} \in \{(1,1),(0,-1),(1,-1),(\pi,1)\}$$

Computing the different trajectories using a Runge-Kutta scheme, we obtain Figure 5.12. The reference $\mathbf{a}(t)$ is painted black when the other solutions found after the action of $\mathbf{g_p}$ are painted red if there was an error during the computation of the trajectory or in another colour if there was no problem. Indeed when calculating the trajectories obtained with the action of $\mathbf{g_p}$, some points could not be computed for some t due to the fact that the quantity $\mathbf{g_p}(\mathbf{a}(t))$ was not defined because of the square root. From Figure 5.12, we have a hint that it may be related to the fact that the initial condition of the transformed trajectory is out of the unit circle. Hence singularities appear when:

$$p_2 + (a_1^2 + a_2^2)(1 - p_2) \leq 0 \tag{5.30}$$

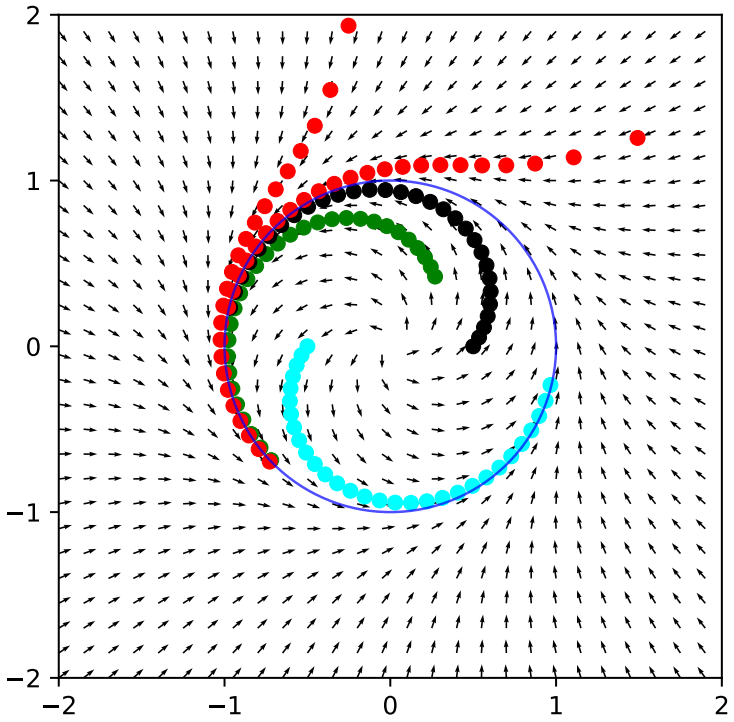


Figure 5.12: Computation of several trajectories. The reference is painted in black, when the trajectories computed using the action $\mathbf{g_p}$ using different values of $\mathbf{p}$ are in colour. Trajectories where there was some error for some points due to the square root are painted red.

5.5.3 The transport function

Now that we have two independent symmetries, following the steps presented in Section 5.2, we need to find the transport function. To do this we use the following:

$$\mathbf{g_p}(\mathbf{a}) = \mathbf{x}.$$

However, in Example 5.5 we noticed that the transformation was not always defined due to the quantity in the square root. Thus we need to find the singularities to check in which case, the symmetries are *complete*. By complete, we mean:

$$\forall \mathbf{x}, \forall \mathbf{a} \in \mathbb{R}^2, \exists \mathbf{p} \in \mathbb{R}^2, \mathbf{x} = \mathbf{g_p}(\mathbf{a}). \tag{5.31}$$

If the symmetries are complete, we can move from any point of the state space to any other point using the symmetry. Otherwise, the pair $(\mathbf{x}, \mathbf{a})$ may be singular.

Proposition 5.3. *Given the symmetries from Equation (5.29) and the system described in Equation (5.23), the pair* $(\mathbf{x}, \mathbf{a})$ *is singular if* $\|\mathbf{a}\| = 1$ *or* $\|\mathbf{x}\| = 0$.

Proof. We would like to check for which pair $(\mathbf{x}, \mathbf{a})$, we have:

$$\exists \mathbf{p} | \mathbf{x} = \mathbf{g}_{\mathbf{p}}(\mathbf{a}).$$

It is trivial to check the existence of p_1, we need to verify the existence of p_2 due to the square root. Thus, we need to verify

$$\exists p_2 |\, \|\mathbf{x}\| = \frac{1}{\sqrt{\|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2}} \|\mathbf{a}\|$$

This is equivalent to

$$\begin{aligned}
\exists p_2 |\, \|\mathbf{x}\| = \frac{1}{\sqrt{\|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2}} \|\mathbf{a}\| \quad &\iff \quad \exists p_2 | \sqrt{\|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2} \cdot \|\mathbf{x}\| = \|\mathbf{a}\| \\
&\iff \quad \exists p_2 | \begin{cases} \|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2 \cdot \|\mathbf{x}\|^2 = \|\mathbf{a}\|^2 \\ \|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2 > 0 \end{cases} \\
&\iff \quad \exists p_2 | \begin{cases} p_2 = \frac{\|\mathbf{a}\|^2 - \|\mathbf{a}\|^2 \cdot \|\mathbf{x}\|^2}{(1 - \|\mathbf{a}\|^2) \cdot \|\mathbf{x}\|^2} \\ \|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) p_2 > 0 \end{cases} \\
&\iff \quad \|\mathbf{a}\|^2 + (1 - \|\mathbf{a}\|^2) \frac{\|\mathbf{a}\|^2 - \|\mathbf{a}\|^2 \cdot \|\mathbf{x}\|^2}{(1 - \|\mathbf{a}\|^2) \cdot \|\mathbf{x}\|^2} > 0,
\end{aligned}$$

which is always true except for $\|\mathbf{a}\| = 1$ or $\|\mathbf{x}\| = 0$. If the pair is not singular, p_2 is given by:

$$p_2 = \frac{\|\mathbf{a}\|^2 - \|\mathbf{a}\|^2 \cdot \|\mathbf{x}\|^2}{(1 - \|\mathbf{a}\|^2) \cdot \|\mathbf{x}\|^2}$$

□

Now that we have characterised the singularities, we can move on to the transport function. Indeed, we know that if we make sure that $\|\mathbf{a}_0\| \neq 1$ and that $\|\mathbf{x}_0\| \neq 0 \iff \mathbf{x}_0 \neq 0$ the transport function will be defined. In that case we already have

$$p_2 = \frac{\|\mathbf{a}\|^2 - \|\mathbf{a}\|^2 \cdot \|\mathbf{x}\|^2}{(1 - \|\mathbf{a}\|^2) \cdot \|\mathbf{x}\|^2}.$$

We now need to compute p_1:

$$\begin{aligned}
\mathbf{g}_{p_1}(\mathbf{a}) = \mathbf{x} \quad &\iff \quad \mathbf{R}_{p_1}(\mathbf{a}) = \mathbf{x} \\
&\iff \quad \begin{pmatrix} \cos p_1 \cdot a_1 - \sin p_1 \cdot a_2 \\ \cos p_1 \cdot a_2 + \sin p_1 \cdot a_1 \end{pmatrix} = \mathbf{x} \\
&\iff \quad p_1 = \operatorname{atan2}(a_1 x_2 - a_2 x_1, a_1 x_1 + a_2 x_2)
\end{aligned}$$

Finally,

$$\mathbf{h}(\mathbf{x}, \mathbf{a}) = \begin{pmatrix} \operatorname{atan2}(a_1 x_2 - a_2 x_1, a_1 x_1 + a_2 x_2) \\ \frac{1}{1 - \|\mathbf{a}\|^2} \left(\frac{\|\mathbf{a}\|^2}{\|\mathbf{x}\|^2} - \|\mathbf{a}\|^2 \right) \end{pmatrix}. \tag{5.32}$$

5.5.4 The flow function

Before getting the analytic function of $\mathbf{\Phi}_t(x)$ in function of $\mathbf{a}(t)$, we need to choose an initial condition for our reference trajectory $\mathbf{a}(\cdot)$. We already have one constraint. Indeed, from our previous work on singularities, we know that $\|\mathbf{a}_0\|$ must not be 1. We tried to integrate the system backward in time, as we will need to do, for different $\mathbf{a}_0$. When doing so, we noticed that when $\mathbf{a}_0$ was outside of the unit disk we systematically encountered an error at some time $t = t_{error} < 0$. But the value of t_{error} seems to be dependent on the value of $\mathbf{a_0}$. We thus need to investigate the problem in order to find a suitable initial condition for our reference.

5.5.4.1 Divergence in a finite time

Finite time divergence is a common concept in the field of ODEs. One classical example to illustrate it is to use the state equation $\dot{x} = x^2$. A solution for this equation is

$$x(t) = \frac{1}{\frac{1}{x_0} - t},$$

where x_0 is the initial condition. This diverges in a finite time. Indeed, when $t = t_\infty = \frac{1}{x_0}$ the system as reached infinity. The flow of the system is given by:

$$\Phi(x_0, t) = \frac{1}{\frac{1}{x_0} - t},$$

and its domain is then

$$\text{dom}\Phi = \{(x_0, t) | t < \frac{1}{x_0}\}.$$

This phenomenon may occur when the system is locally but not globally Lipschitz as it is the case for the system considered in TC3.

5.5.4.2 Backward divergence of the system of test case 3

We will show here that the system diverges backward in a finite time. If we define $r = \sqrt{x_1^2 + x_2^2}$ we get from the work done in Section 5.5.2, $\dot{r} = -r^3 + r$. A solution for this ODE is

$$r(t) = \frac{e^t}{\sqrt{\frac{1}{r_0^2} - 1 + e^{2t}}}$$

This mean that the divergence is obtained for time t_∞ which satisfies

$$\begin{aligned} & \frac{1}{r_0^2} - 1 + e^{2t_\infty} = 0 \\ \iff \quad & t_\infty = \frac{1}{2} \log\left(1 - \frac{1}{r_0^2}\right) \end{aligned}$$

Hence, the finite time divergence is obtained only if $1 - \frac{1}{r_0^2} > 0$ *i.e* when $r_0^2 > 1$ which means, when the system is initialised outside of the unit circle. This is illustrated in Figure 5.13.

The arrows shows the direction of the integration backward. We can see that for the purple and orange curves are diverging towards infinite. And the numerical integration scheme quickly throws an error. In this case $t_\infty < 0$. Thus the finite divergence appears when we integrate backward in time. In the end, to be certain that we will be able to compute the reference correctly, we choose $\mathbf{a}_0$ such that it belongs to the inside of the unit disk.

5.5.4.3 Getting the flow function

From both constraints, we conclude that in order to compute the flow, we should choose a value of $\mathbf{a}_0$ for which $\|\mathbf{a}_0\| < 1$ in order to avoid backward finite divergence and singularities. Here we choose $\mathbf{a}_0 = (\frac{1}{2}, 0)^\intercal$. It obviously satisfies, $\|\mathbf{a}_0\| < 1$. From this we have:

$$\begin{aligned}\mathbf{\Phi}_t(\mathbf{x}) &= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{a}_0)} \circ \mathbf{a}(t) \\ &= \frac{\begin{pmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{pmatrix} \begin{pmatrix} a_1^0 & a_2^0 \\ -a_2^0 & a_1^0 \end{pmatrix} \cdot \mathbf{a}(t)}{\|\mathbf{a}_0\|^2 \sqrt{\frac{1-\|\mathbf{a}(t)\|^2}{1-\|\mathbf{a}_0\|^2} + \left(\frac{\|\mathbf{a}(t)\|^2}{\|\mathbf{a}_0\|^2} - \frac{1-\|\mathbf{a}(t)\|^2}{1-\|\mathbf{a}_0\|^2}\right)\|\mathbf{x}\|^2}}.\end{aligned}$$

Hence for $\mathbf{a}_0 = (\frac{1}{2}, 0)^\intercal$:

$$\mathbf{\Phi}_t(\mathbf{x}) = \frac{\sqrt{3}\begin{pmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{pmatrix} \cdot \mathbf{a}(t)}{\sqrt{1 - \|\mathbf{a}(t)\|^2 + (4\|\mathbf{a}(t)\|^2 - 1)\|\mathbf{x}\|^2}}. \tag{5.33}$$

5.5.5 Solving the constraint satisfaction problem

To illustrate and compare our integration method, we will use the example considered in [24], where the initial set is a disk with centre (1.5,1.5) and a radius 0.2. We will denote this set by $\mathbb{D} = \{\mathbf{x} | (x_1 - 1.5)^2 + (x_2 - 1.5)^2 \leq 0.2^2\}$. For a defined time t, we want to characterise the set

$$\mathbb{X}_t = \{\mathbf{x} | \mathbf{\Phi}_{-t}(\mathbf{x}) \in \mathbb{D}\}. \tag{5.34}$$

To approximate the inside of $\mathbb{X}_t$ using our contractor-based approach, we need to characterise $\overline{\mathbb{X}_t}$ also. We have

$$\overline{\mathbb{X}_t} = \{\mathbf{x} | \mathbf{\Phi}_{-t}(\mathbf{x}) \notin \mathbb{D} \text{ or } (\mathbf{x}, t) \notin \text{dom}\mathbf{\Phi}\}. \tag{5.35}$$

One should notice that $\mathbf{\Phi}_{-t}$ is not always defined, $(\mathbf{x}, t) \in \text{dom}\mathbf{\Phi}$ may be false for some $t < 0$.

5.5.5.1 Approximation using CAPD to compute the reference

We performed the integration for multiple sets $\mathbb{X}_t, t \in T = \{0, 0.1, 0.25, 0.5, 0.75, 1, 2, 3, 4, 5, 6\}$. The result is shown in Figure 5.14. As in previous test cases, the inner approximations are

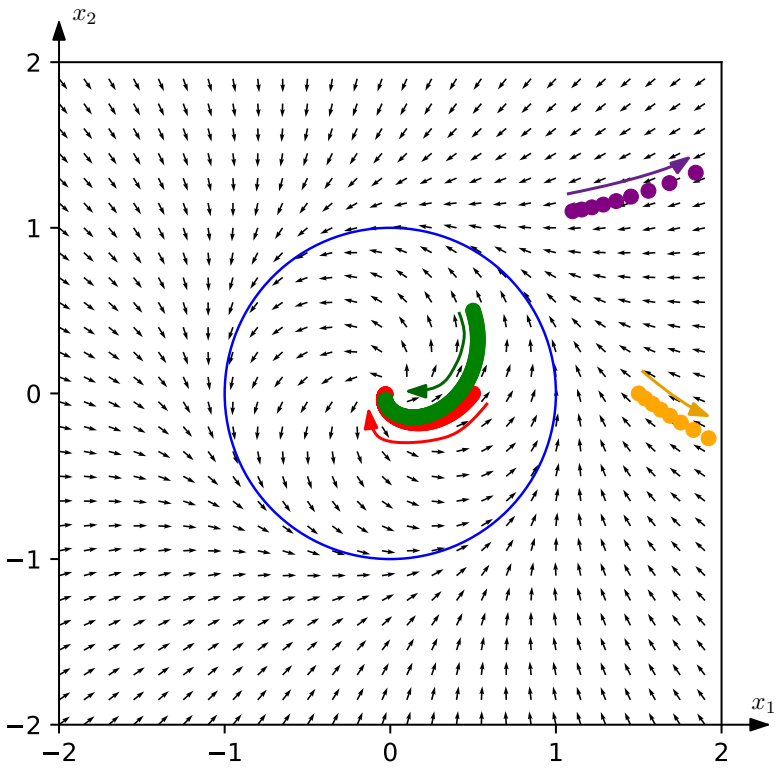


Figure 5.13: Finite divergence when integrating backward.

painted white when the outer approximations are painted blue. The boundary is still in pink. The code used to compute the approximations can be found here[2]. As, for now, we do not have an analytical expression for $\mathbf{a}(t)$, a sharp enclosure of it is computed as a tube using a conventional guaranteed integration tool (here CAPD) and stored in a tube. The reader of the example code may have noticed that to build the separator, a new object, the ContractorNetwork, has been used. This is, for now, a workaround for IBEX inability to compute with tubes but not its intended use. This issue will be solved in a future release of Codac. This new tool will be presented in Chapter 6. As already done in TC1 and TC2, we can compute the forward reach set. This time we will do it for $t \in [0, t_{max} = 6]$.

$$\mathbb{X}_t = \{\mathbf{x}, \exists t \in [0, t_{max}] | \mathbf{\Phi}_{-t}(\mathbf{x}) \in \mathbb{D}\} \tag{5.36}$$

and

$$\overline{\mathbb{X}_t} = \{\mathbf{x}, \exists t \in [0, t_{max}] | \mathbf{\Phi}_{-t}(\mathbf{x}) \notin \mathbb{D} \text{ or } (\mathbf{x}, t) \notin \text{dom}\mathbf{\Phi}\}$$

The result of the computation is shown in Figure 5.15. A summary of the computations results is available in Table 5.8 at the end of the section.

[2]TODO

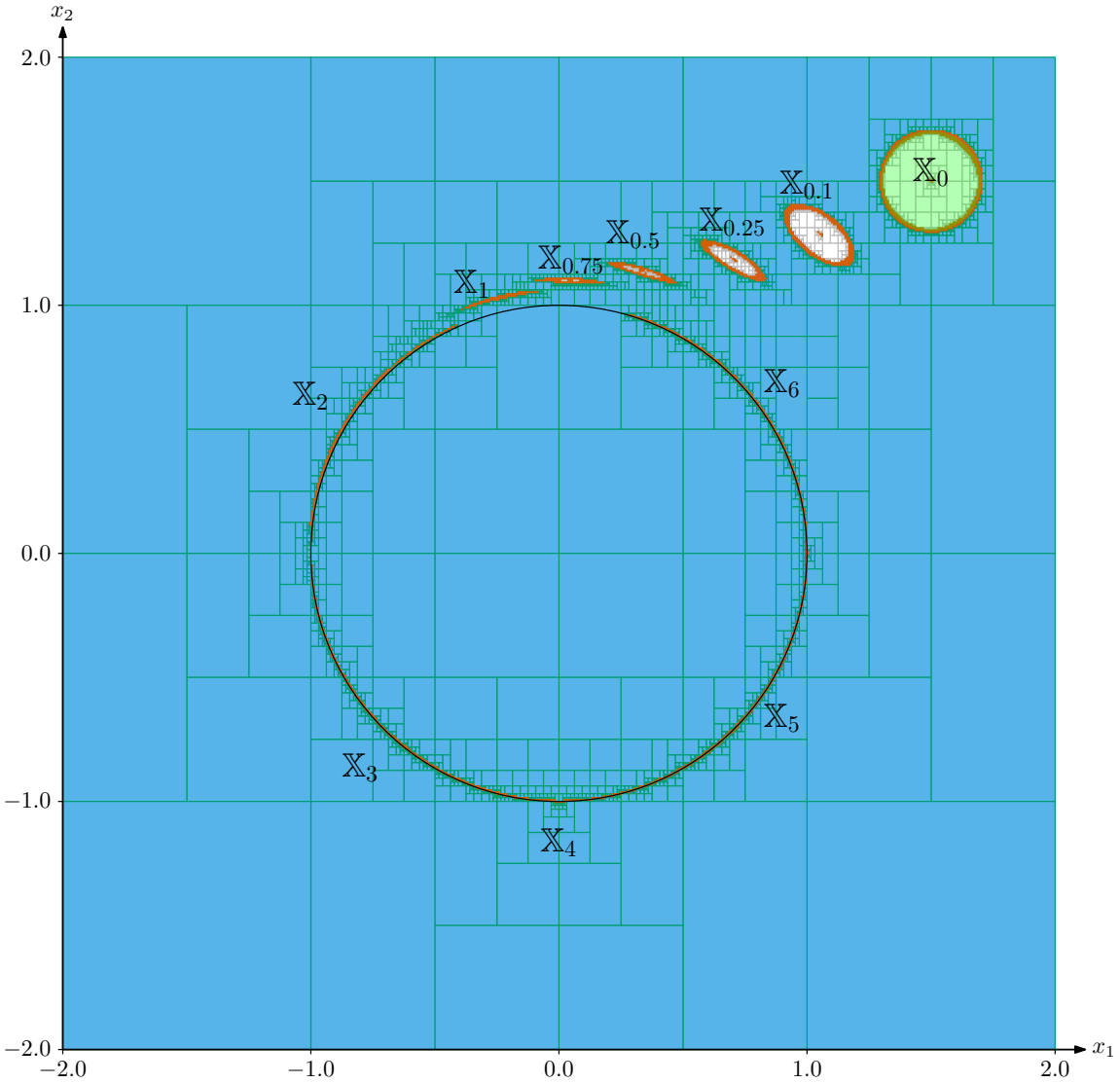


Figure 5.14: Integration of multiple discrete sets for TC3, using CAPD to compute the reference.

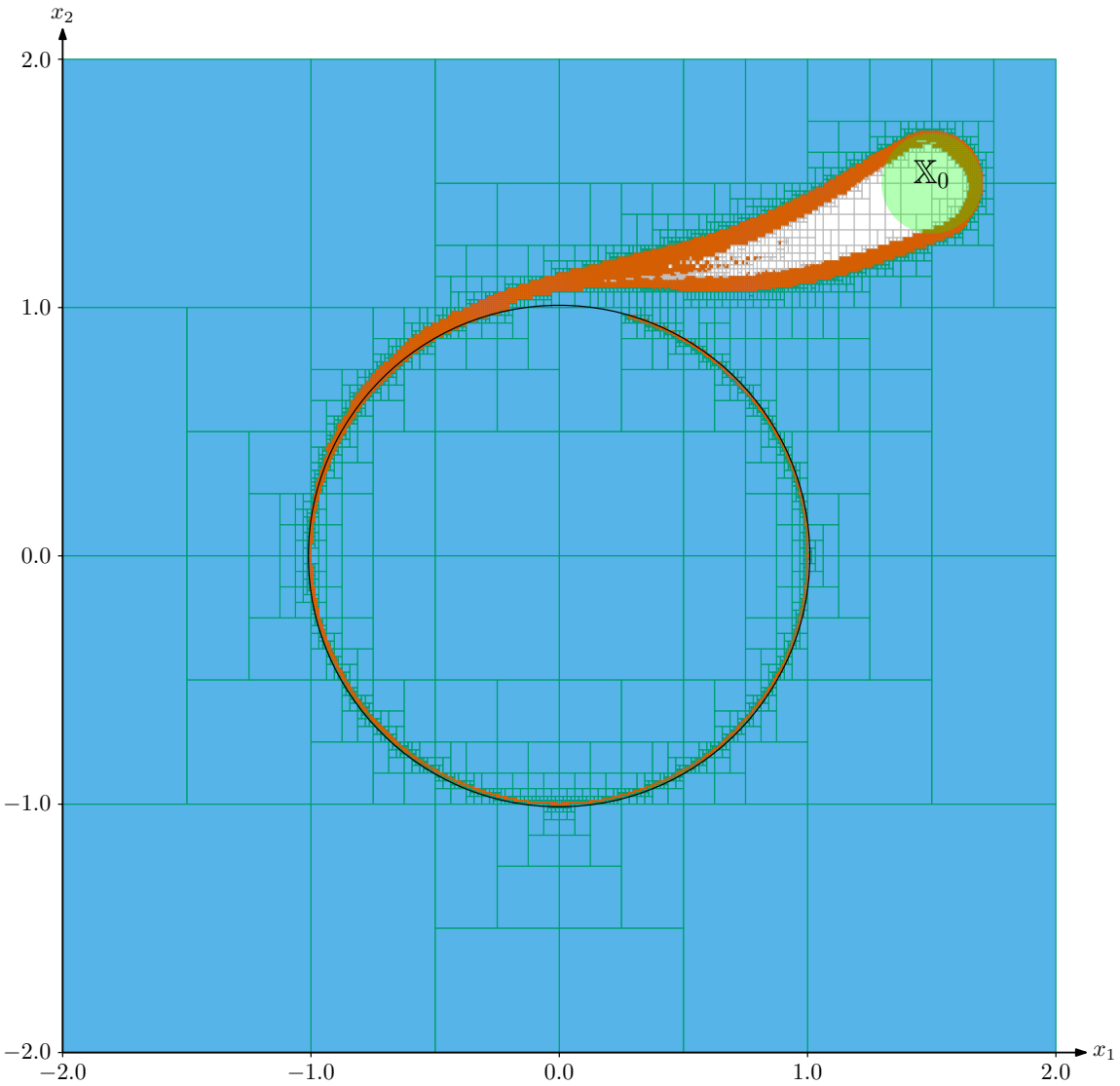


Figure 5.15: Continuous integration applied on TC3 using CAPD to compute the reference. The thickness of the frontier is due to two factors. The first one is the fact that we perform a guaranteed integration to compute the reference instead of using an analytical expression which adds pessimism on the reference. And thus reduces the volume of the inner approximation. This is solved in Section 5.5.5.2 The second one is the use of projection algorithm over time.

5.5.5.2 Approximation using the analytical expression of the reference

It is actually possible to get an analytical expression for the reference $\mathbf{a}(t)$. From the expression we have for $\dot{\theta}$ and $\dot{r}$ in Section 5.5.2, we obtain

$$\begin{aligned} \theta(t) &= \theta(0) + t \\ r(t) &= \frac{e^t}{\sqrt{\frac{1}{r_0^2} - 1 + e^{2t}}}. \end{aligned}$$

Hence

$$\mathbf{\Phi}(t, \mathbf{x}(0)) = \frac{e^t}{\sqrt{1 + (e^{2t} - 1) \left\|\mathbf{x}(0)\right\|^2}} \begin{pmatrix} cc \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix} \cdot \mathbf{x}(0). \tag{5.37}$$

With our initial condition $\mathbf{a}_0 = (\frac{1}{2}, 0)^\intercal$, we thus have:

$$\mathbf{a}(t) = \frac{1}{\sqrt{3e^{-2t} + 1}} \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}. \tag{5.38}$$

With this formulation for the reference, we obtain similar results as the previous formulation but 26 times faster in the continuous case and more than 200 times in the discrete case. The results of computation are displayed in Appendix A, in Figure A.1 and Figure A.2 This was expected as getting a value from a tube object at a defined interval of time $[t]$ takes longer than computing the value of the analytic expression of the reference for this interval of time. Moreover as shown in Table 5.8, we need less bisections as the enclosure of the reference is sharper because we are not performing integration but analytic computation to get it. This leads to a significant improvement in terms of processing time.

Experiment:				
Reference tool	continuous		discrete	
	nb bisections	processing time	nb bisections	processing time
CAPD	8631	2315 s	4584	964 s
analytic expression	6737	90.8 s	4036	4.5 s

Table 5.8: Summary of TC3 computations

5.6 A robotic test case

5.6.1 Presentation of test case 4

To conclude this chapter, we will consider one last test case denoted Test-Case 4 (TC4), coming from robotics. The system we are going to consider is called the Dubins' car [32, 60] and is defined by the following state equation:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}(t)) = \begin{pmatrix} u_1(t) \cdot \cos x_3 \\ u_1(t) \cdot \sin x_3 \\ u_2(t) \end{pmatrix}, \tag{5.39}$$

where u_1 and u_2 are both time dependent. In order to make the system autonomous and avoid the time dependency in $\mathbf{u}$, we can rewrite the system into:

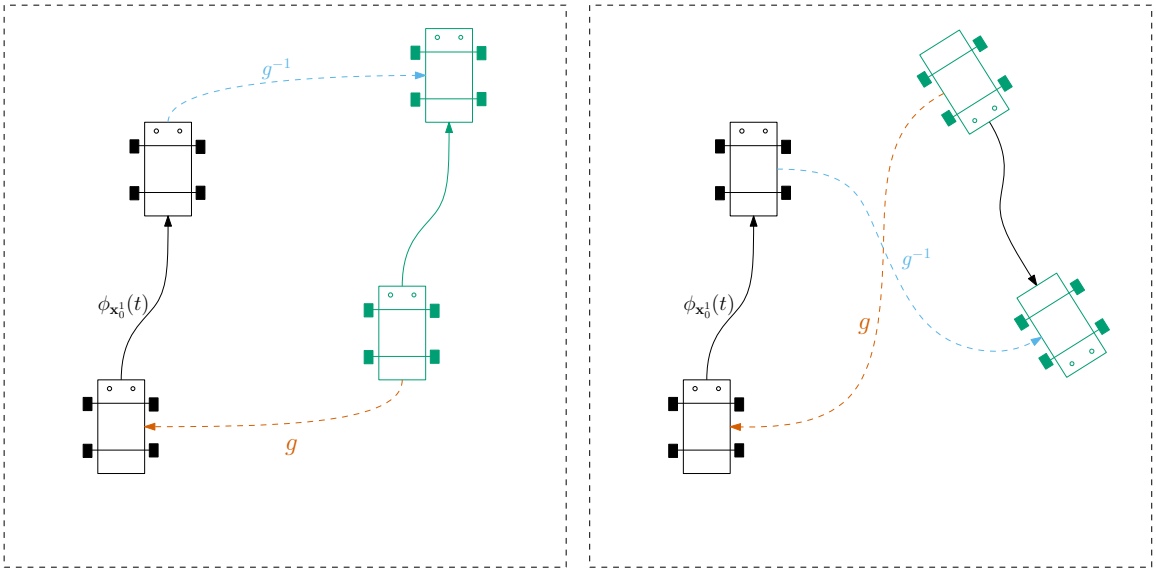
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} u_1(x_4) \cdot \cos x_3 \\ u_1(x_4) \cdot \sin x_3 \\ u_2(x_4) \\ 1 \end{pmatrix}, \tag{5.40}$$

where x_4 will denote the clock variable.

5.6.2 Finding symmetries

Due to the fact that our system depends on $\mathbf{u}$, the vector field associated with $\mathbf{f}$ varies when $\mathbf{u}$ evolves thus we will not use it to intuitively find the symmetries. Instead we will use the other meaning of $\mathbf{f}$. Indeed, $\mathbf{f}$ describes how the system evolves through time. Taking back the example used in Section 4.2, we can easily determine the symmetries with intuition. We recall the operation done in Section 4.2 in Figure 5.16a.

Consider now that our black character is a Dubins car. If there is no perturbation whatsoever, the path followed by our green car, which is a copy of the black one, should be the same, only the initial condition changes. And we have here two translation symmetries, along Ox_1 and Ox_2. Now let us suppose that our green car is, at his initial position, rotated sixty degrees to the right. Again, if it does not come across any disturbances, our green car will follow the same path rotated by ninety degree (see Figure 5.16b). This gives us one



(a) Translation symmetries along Ox_1 and Ox_2 (b) Rotation symmetry illustration

Figure 5.16: Available spatial symmetries for the Dubins car

rotation symmetry. We already have three independent symmetries for the first 3 degrees of freedom x_1, x_2 and x_3. The last symmetry we will use is the one parameter flow symmetry, presented in Example 4.9 and used in TC2. This leads us to the following proposition.

Proposition 5.4. *The transformations*

$$\mathbf{g_p}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \mathbf{R}_{p_3} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ x_3 + p_3 \\ x_4 \end{pmatrix} \circ \mathbf{\Phi}_{p_4}(\mathbf{x}) \tag{5.41}$$

where $\mathbf{R}_{p_3}$ is the rotation matrix in 2D, are Lie symmetries for the system defined in Equation (5.40).

This transformation corresponds to the translation along Ox_1 and Ox_2 (parameterised by p_1 and p_2 respectively) and the rotation (parameterised by p_3). The three first components of the symmetry $\mathbf{g_p}$ corresponds to the direct Euclidean group $SE(2)$ (see Example 4.5).

Proof. We have already seen that the one parameter flow symmetry is a symmetry. It is denoted by $\mathbf{\Phi}_{p_4}$. Let us now introduce:

$$\mathbf{g}_{p_1}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} p_1 + x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \tag{5.42}$$

$$\mathbf{g}_{p_2}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ p_2 + x_2 \\ x_3 \\ x_4 \end{pmatrix}, \tag{5.43}$$

and

$$\mathbf{g}_{p_3}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{p_3} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ p_3 + x_3 \\ x_4 \end{pmatrix}. \tag{5.44}$$

Firstly, let us check that $\mathbf{g}_{p_1}$ is a symmetry. On the one hand we have:

$$\left(\frac{d\mathbf{g}_{p_1}}{d\mathbf{x}}\right) \cdot \mathbf{f}(\mathbf{x}) = \mathbf{I}_{4\times 4} \cdot \mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}).$$

And on the other hand:

$$\mathbf{f}(\mathbf{x}) \circ \mathbf{g}_{p_1}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \circ \begin{pmatrix} p_1 + x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \mathbf{f}(\mathbf{x})$$

Hence:

$$\left(\frac{d\mathbf{g}_{p_1}}{d\mathbf{x}}\right) \cdot \mathbf{f} = \mathbf{f} \circ \mathbf{g}_{p_1}.$$

From the equivariance property, we can conclude that $\mathbf{g}_{p_1}$ is a symmetry for Equation (5.40). It corresponds to the translation along the Ox_1 axis. In the exact same way we can show that $\mathbf{g}_{p_2}$ is also a symmetry corresponding to the translation along the Ox_2 axis.
Finally,

$$\begin{aligned} \left(\frac{d\mathbf{g}_{p_3}}{d\mathbf{x}}\right) \cdot \mathbf{f}(\mathbf{x}) &= \begin{pmatrix} \mathbf{R}_{p_3} & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{I}_{2\times 2} \end{pmatrix} \cdot \begin{pmatrix} u_1(x_4) \cdot \cos x_3 \\ u_1(x_4) \cdot \sin x_3 \\ u_2(x_4) \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} u_1(x_4) \cdot \cos(x_3 + p_3) \\ u_1(x_4) \cdot \sin(x_3 + p_3) \\ u_2(x_4) \\ 1 \end{pmatrix}, \end{aligned}$$

and

$$\begin{aligned}\mathbf{f}\circ\mathbf{g}_{p_3} &= \begin{pmatrix} u_1(x_4)\cdot\cos x_3 \\ u_1(x_4)\cdot\sin x_3 \\ u_2(x_4) \\ 1 \end{pmatrix} \circ \begin{pmatrix} \cos p_3\cdot x_1 - \sin p_3\cdot x_2 \\ \sin p_3\cdot x_1 + \cos p_3\cdot x_2 \\ x_3+p_3 \\ x_4 \end{pmatrix} \\ &= \begin{pmatrix} u_1(x_4)\cdot\cos(x_3+p_3) \\ u_1(x_4)\cdot\sin(x_3+p_3) \\ u_2(x_4) \\ 1 \end{pmatrix}.\end{aligned}$$

Again, with the equivariance property, we get that $\mathbf{g}_{p_3}$ is a symmetry. Thus, as $\mathbf{g_p} = \mathbf{g}_{p_1}\circ\mathbf{g}_{p_2}\circ\mathbf{g}_{p_3}\circ\mathbf{\Phi}_{p_4}$, $\mathbf{g_p}$ is a stabiliser as a composition of stabilisers. □

5.6.3 The transport function

Now that we have complete symmetries, it is possible to compute the transport function. To do this, we will once again use

$$\mathbf{g_p}(\mathbf{a}) = \mathbf{x},$$

to obtain an equation of the form

$$\mathbf{p} = \mathbf{h}(\mathbf{x},\mathbf{a}).$$

Hence

$$\mathbf{g_p} = \mathbf{h}(\mathbf{x},\mathbf{a}) \iff \begin{cases} \begin{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \mathbf{R}_{p_3}\cdot\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ y_3+p_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \\ \mathbf{y} = \mathbf{\Phi}_{p_4}(\mathbf{a}) \end{cases}. \tag{5.45}$$

Now, from Section 5.6.3 we have $x_4 = y_4 = \phi_{p_4}(\mathbf{a}) = a_4 + p_4$ as x_4 corresponds to the system clock. Thus

$$p_4 = x_4 - a_4.$$

Therefore,

$$\begin{aligned}\mathbf{g_p} = \mathbf{h}(\mathbf{x},\mathbf{a}) &\iff \begin{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \mathbf{R}_{p_3}\cdot\begin{pmatrix} \phi_{p_4,1}(\mathbf{a}) \\ \phi_{p_4,2}(\mathbf{a}) \end{pmatrix} \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3-\phi_{p_4,3}(\mathbf{a}) \\ x_4-a_4 \end{pmatrix} \\ &\iff \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \mathbf{x} - \begin{pmatrix} \mathbf{R}_{x_3-\phi_{x_4-a_4,3}(\mathbf{a})}\cdot\begin{pmatrix} \phi_{x_4-a_4,1}(\mathbf{a}) \\ \phi_{x_4-a_4,2}(\mathbf{a}) \end{pmatrix} \\ \phi_{x_4-a_4,3}(\mathbf{a}) \\ a_4 \end{pmatrix} = \mathbf{h}(\mathbf{x},\mathbf{a}).\end{aligned}$$

5.6.4 The flow function

We have done most of the preliminary work. It remains to choose an initial condition for our reference. Here we choose $\mathbf{a}_0 = (0,0,0,0)^\intercal$. We can now compute the transport function in our particular case. We have $\mathbf{\Phi}_t(\mathbf{0}) = \mathbf{a}(t)$ (no displacement from the reference), hence, $\mathbf{\Phi}_{x_4}(\mathbf{0}) = \mathbf{a}_{x_4}$. Thus:

$$\mathbf{h}(\mathbf{x},\mathbf{0}) = \mathbf{x} - \begin{pmatrix} \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} a_1(x_4) \\ a_2(x_4) \end{pmatrix} \\ a_3(x_4) \\ 0 \end{pmatrix}. \tag{5.46}$$

Moreover,

$$\begin{aligned} \mathbf{g_p} \circ \mathbf{a}(t) &= \begin{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \mathbf{R}_{p_3} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ x_3 + p_3 \\ x_4 \end{pmatrix} \circ \underbrace{\mathbf{\Phi}_{p_4}(\mathbf{a}(t))}_{\mathbf{a}(t+p_4)} \\ &= \begin{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \mathbf{R}_{p_3} \cdot \begin{pmatrix} a_1(t+p_4) \\ a_2(t+p_4) \end{pmatrix} \\ a_3(t+p_4) + p_3 \\ a_4(t+p_4) \end{pmatrix}. \end{aligned} \tag{5.47}$$

Combining Equation (5.46) and Equation (5.47), we obtain:

$$\begin{aligned} \mathbf{\Phi}_t(\mathbf{x}) &= \mathbf{g}_{\mathbf{h}(\mathbf{x},\mathbf{0})} \circ \mathbf{a}(t) \\ &= \begin{pmatrix} \begin{pmatrix} h_1(\mathbf{x},\mathbf{0}) \\ h_2(\mathbf{x},\mathbf{0}) \end{pmatrix} + \mathbf{R}_{h_3(\mathbf{x},\mathbf{0})} \cdot \begin{pmatrix} a_1(t + h_4(\mathbf{x},\mathbf{0})) \\ a_2(t + h_4(\mathbf{x},\mathbf{0})) \end{pmatrix} \\ a_3(t + h_4(\mathbf{x},\mathbf{0})) + h_3(\mathbf{x},\mathbf{0}) \\ a_4(t + h_4(\mathbf{x},\mathbf{0})) \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} a_1(t+x_4) - a_1(x_4) \\ a_2(t+x_4) - a_2(x_4) \end{pmatrix} \\ x_3 + a_3(t+x_4) + a_3(x_4) \\ a_4(t+x_4) \end{pmatrix}. \end{aligned} \tag{5.48}$$

5.6.5 Applying the Lie integration method on the robotic test case

5.6.5.1 Problem presentation

Now that we have an analytical expression of $\mathbf{\Phi}_t(\mathbf{x})$, we illustrate it on an example. From Equation (5.40), we see that we need an expression for $\mathbf{u}(t)$. We will use the following,

$$\mathbf{u}(t) = \begin{pmatrix} \sin(0.4 * t) \\ 1 \end{pmatrix}, \tag{5.49}$$

where $u_1(t)$ represents the control given for the yaw of the Dubins car and $u_2(t)$ its speed, here set as a constant. We will use as initial condition for our reference, $\mathbf{a}_0 = (0,0,0,0)^\intercal$ and our objective is to compute the trajectory of the system over 15 s. Using CAPD, we

obtain the reference trajectory depicted in Figure 5.17. The beginning of the curve at $t = 0$ is black and the end is in green when $t = 5$.

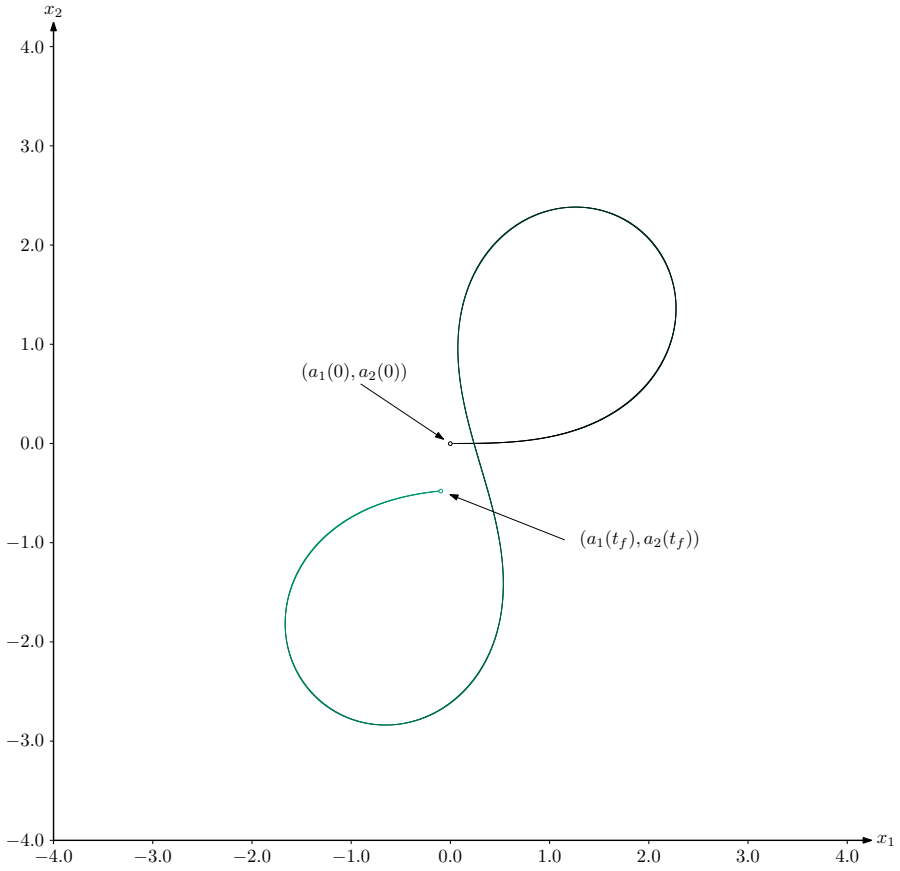


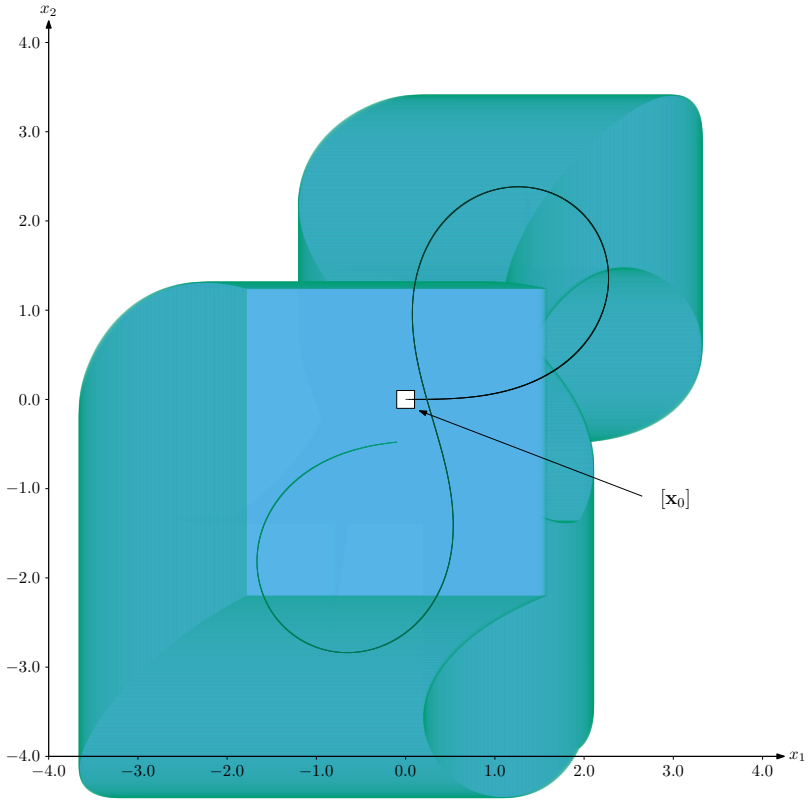
Figure 5.17: Reference trajectory computed using CAPD on TC4.

The uncertain initial condition for which we would like to compute the trajectory is given by $[\mathbf{x}_0] = [-0.1, 0.1]^2 \times [-0.4, 0.4] \times [0, 0]$. We first computed the tubes obtained using CAPD and Flow*. Both results are depicted in Figure 5.18. The tube obtained with CAPD is in blue when the one obtained with Flow* is in red. One can notice that, once again, Flow* is more robust to the uncertain initial condition than CAPD. In terms of processing time, CAPD is faster with 71 ms when Flow* takes 4.2 s to compute its results. But considering the trade-off sharpness of enclosure *vs* processing time, Flow* is the one that performs the best.

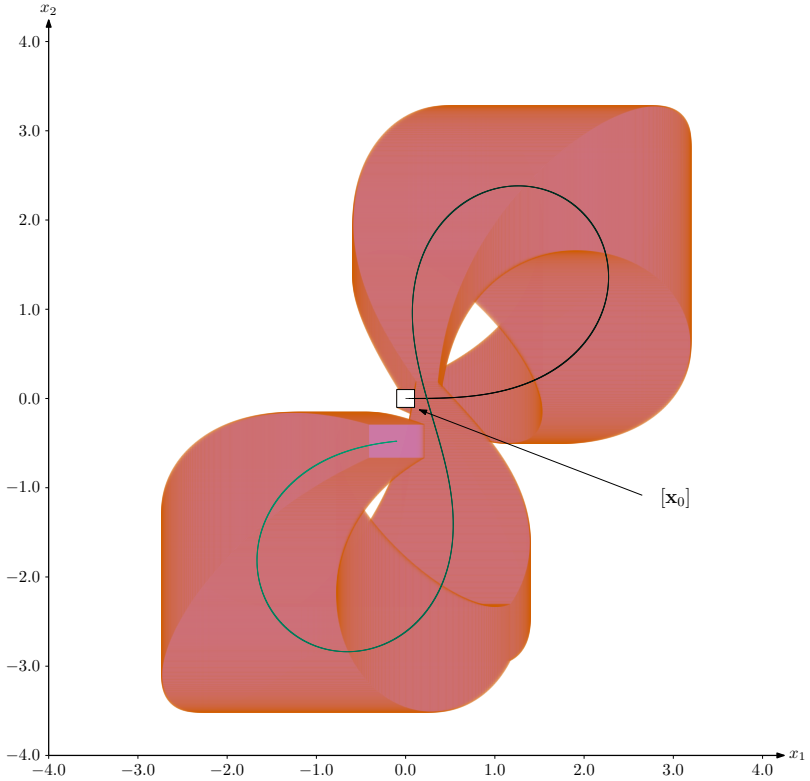
5.6.5.2 Simplification

To compute the tube using the Lie integration method, we will contract each slice $[\mathbf{x}]$ of the tube using the constrain $c_{phi} : \mathbf{\Phi}_{-t}(\mathbf{x}) \in [\mathbf{x}_0]$. Hence, from the last component of Equation (5.48), for $\mathbf{x}$ to be solution, we must have

$$\begin{aligned} a_4(-t + x_4) \in [0, 0] &\iff a_4(-t + x_4) = [0, 0] \\ &\iff x_4 - t = 0 \end{aligned} . \tag{5.50}$$



(a) Integration with CAPD.



(b) Integration with Flow*.

Figure 5.18: Solution obtained for TC4 using CAPD and Flow*.

Therefore it is possible to simplify Equation (5.48) into

$$\begin{aligned}
\boldsymbol{\Phi}_{-t}(\mathbf{x}) &= \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} a_1(x_4 - t) - a_1(x_4) \\ a_2(x_4 - t) - a_2(x_4) \end{pmatrix} \\ x_3 + a_3(x_4 - t) + a_3(x_4) \\ a_4(x_4 - t) \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} a_1(0) - a_1(x_4) \\ a_2(0) - a_2(x_4) \end{pmatrix} \\ x_3 + a_3(0) + a_3(x_4) \\ a_4(0) \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} 0 - a_1(x_4) \\ 0 - a_2(x_4) \end{pmatrix} \\ x_3 + 0 + a_3(x_4) \\ 0 \end{pmatrix}
\end{aligned} , \tag{5.51}$$

and thus obtaining a 3D problem:

$$\boldsymbol{\Phi}_{-t}(\mathbf{x}) = \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{R}_{x_3 - a_3(x_4)} \cdot \begin{pmatrix} -a_1(x_4) \\ -a_2(x_4) \end{pmatrix} \\ x_3 + a_3(x_4) \end{pmatrix} . \tag{5.52}$$

Our initial condition for this 3D problem is $[\mathbf{x}_0] = [-0.1, 0.1]^2 \times [-0.4, 0.4]$. The reason for which we need to simplify this problem is that, in order to get an inner approximation using our method, the constraint set on each parameter, must not be a single value but an interval. Otherwise, as we are computing with intervals, the result obtained with $[x_4] - [t]$ will be an enclosure for the singular set $\{0\}$ but not the set itself, even if $[x_4] = [t]$. Therefore, there could not be any element $[\mathbf{y}] = [\boldsymbol{\Phi}]_{[-t]}([\mathbf{x}])$ such that $[\mathbf{y}] \subset [\mathbf{x}_0]$. With the change made, we obtain the tube painted yellow in Figure 5.19. The lie integration method is, again, more robust than the two other solvers. In terms of processing time, the tube is computed in 3.2 s which is faster than Flow*. Thus Lie integration method is more efficient than the other two solvers on this test case.

Finally, as we did for with the previous test cases, we computed the inner and outer approximation for multiple sets $\mathbb{X}_{t_i}$ with $t_i \in T = \{1, 2, 3, 4, 14, 15\}$ and the full forward reach set $\mathbb{X}_t$ with $t \in [0, 15]$. The results after projection on the x_1 and x_2 coordinates are depicted in Figure 5.20 and Figure 5.21. When comparing the enclosure obtained in Figure 5.21 and the one obtained with the tube in Figure 5.19, we can notice that the SIVIA algorithm improves the finesse of the outer enclosure (white + pink) and of course we can compute easily an inner approximation of the solution set.

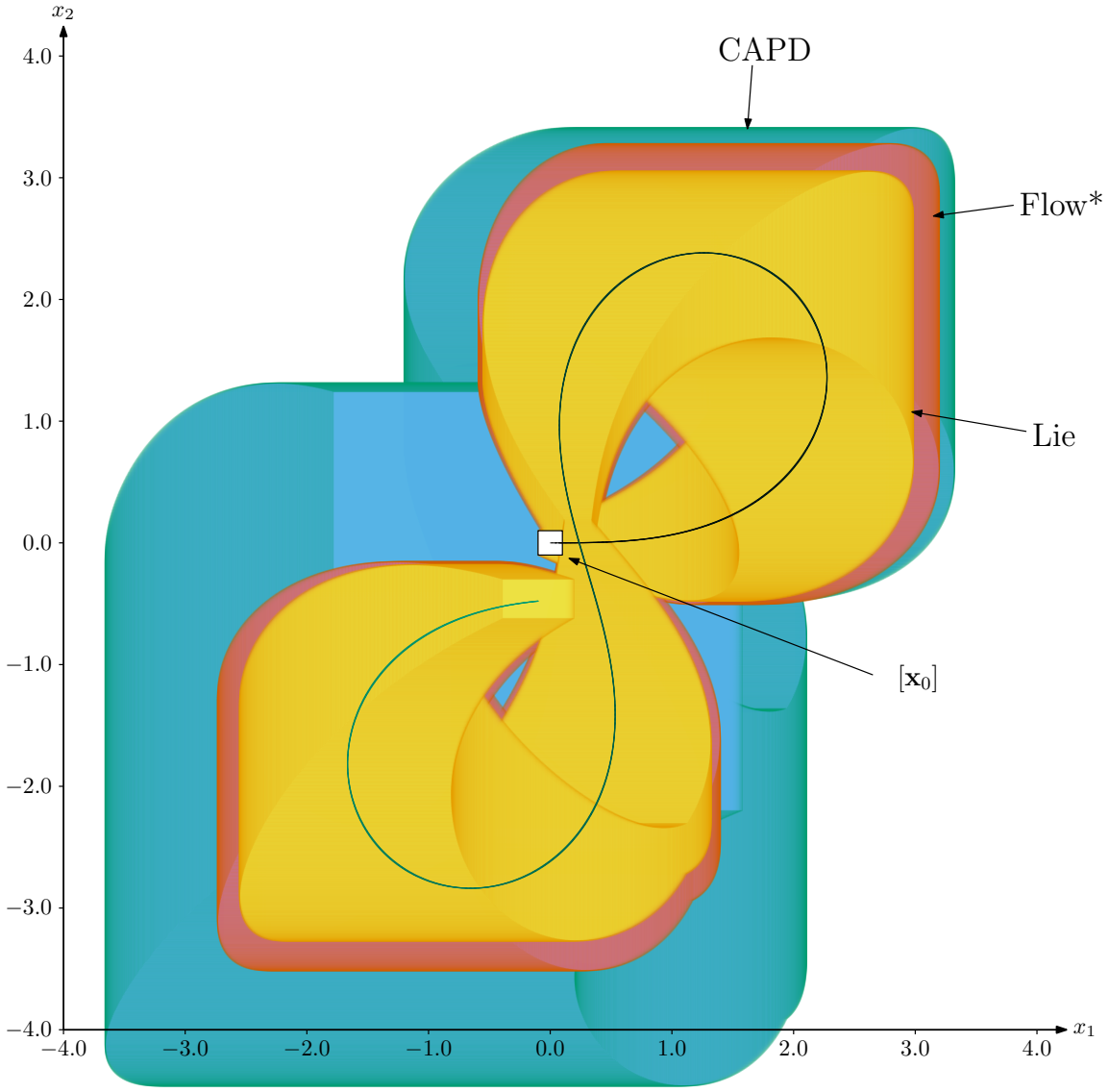


Figure 5.19: Comparison of the tube obtained with the Lie integration method with the results of CAPD and Flow*.

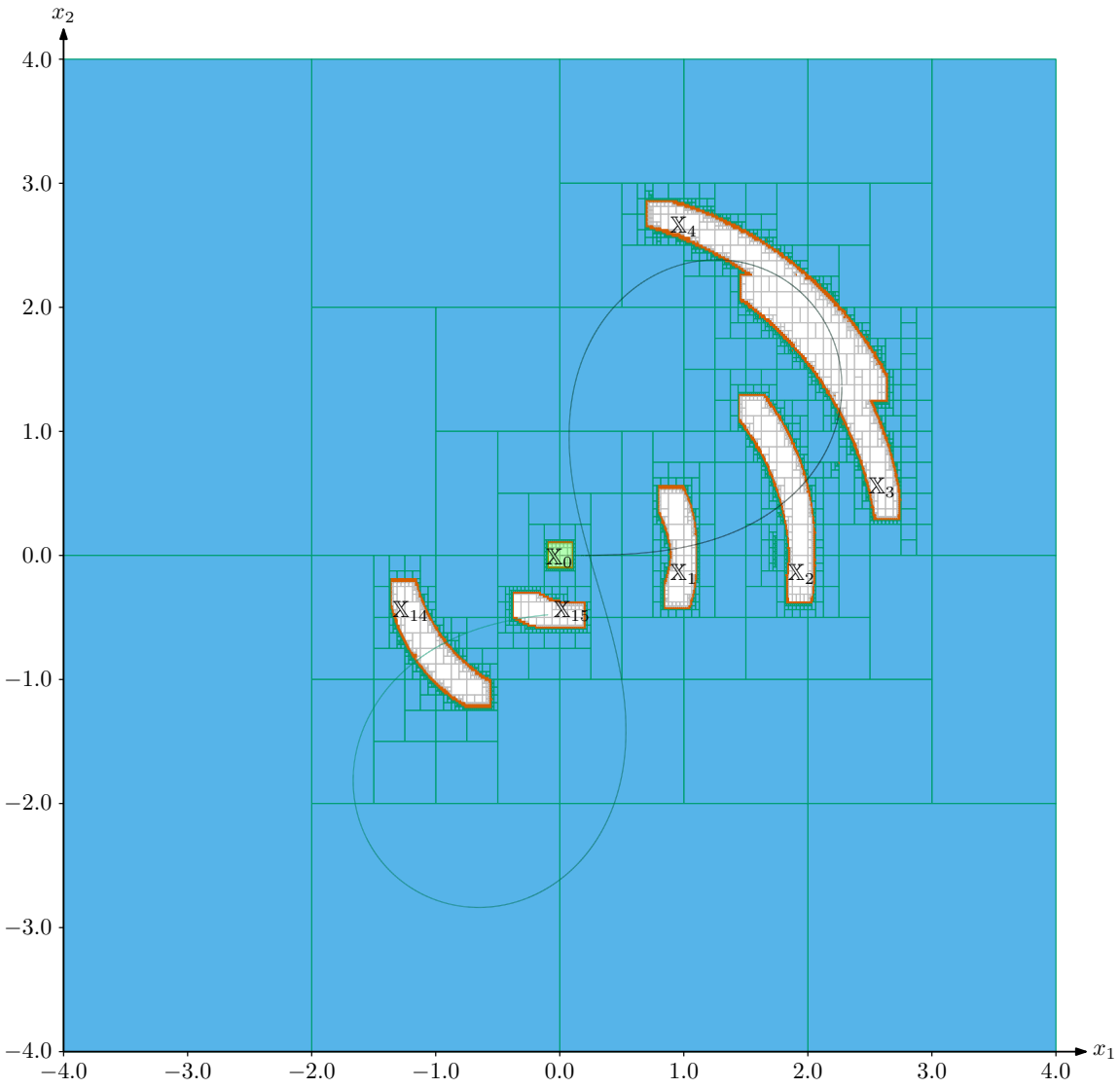


Figure 5.20: Result of TC4 for multiple discrete time sets: $\underset{(x_1,x_2)t_i\in T}{\text{Proj}} \bigcup \mathbb{X}_{t_i}$.

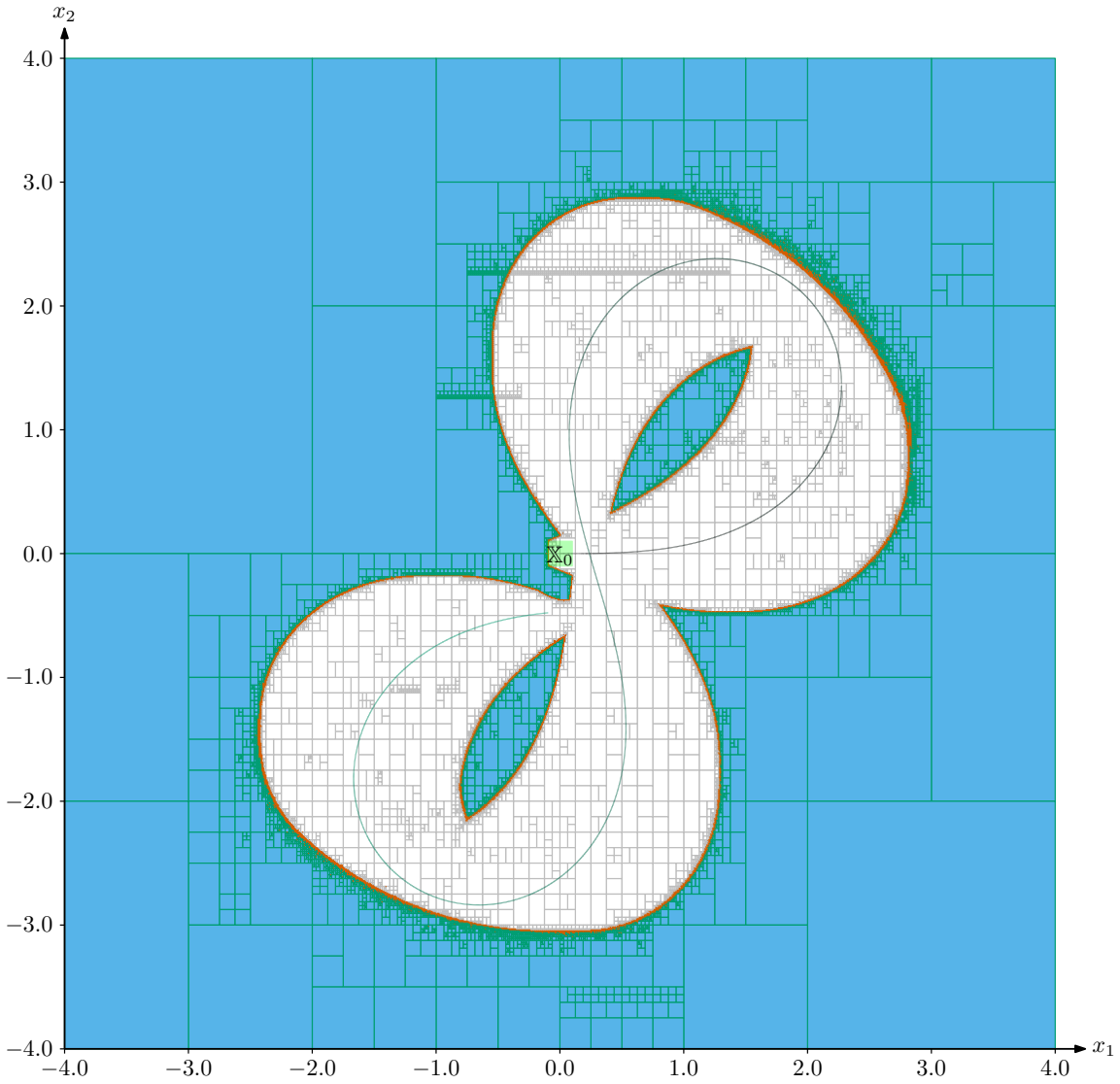


Figure 5.21: Result of TC4 for a continuous time set: $\underset{(x_1,x_2)t\in[0,15]}{\text{Proj}} \bigcup \mathbb{X}_t$.

5.6.5.3 Lawnmower pattern

One may have noticed in the different figures presented throughout this last test case that the volume of the tubes was shrinking when the trajectory was coming back close to the initial condition, no matter which solver was used. This is a bit counter-intuitive if we consider that the error should increase from one step to the following. This phenomenon can be explained by the fact that when the systems comes back to its initial position, the error computed is inverted compared to the one computed when the robot was going away from its initial position. Hence previous errors are compensated on the way back which increases the accuracy of the result. This is demonstrated in Appendix A.2 with the help of Alain Betholom engineer at ENSTA Bretagne. This phenomenon is observed with many inertial systems often equipped with a variation of the Kalman filter as in [84, Figure 6]. This the reason why special patterns have been developed when exploring a zone with a robot equipped with an INS, especially the lawnmower pattern which is depicted in Figure 5.22. This pattern uses the property described above to limit the error increase during the mission, which allows a better estimation of the position by the INS.

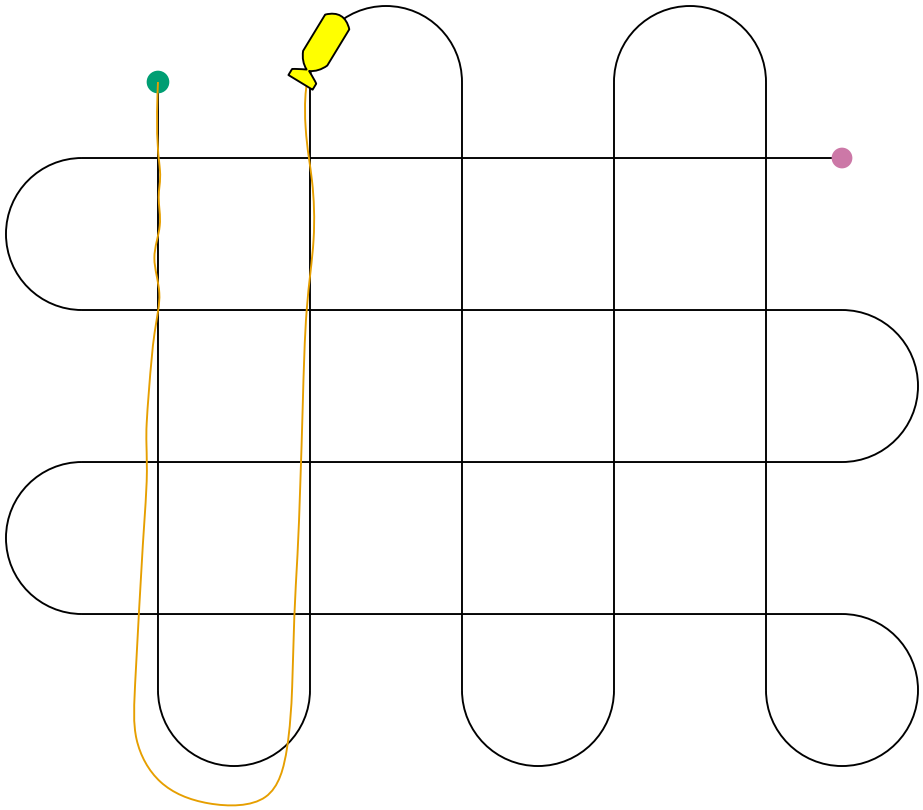


Figure 5.22: The lawnmower pattern: the robot is set to follow the lawnmower pattern from the green point to the pink one following the black trajectory. In orange is depicted the estimation of the position made by the INS. One can notice that this error decreases when the robot is coming closer to its initial position.

5.7 Prospects

Throughout this chapter, we applied the symmetries on the state space, *i.e* when changing the initial condition we were moving along the axes $Ox_1, \ldots, Ox_n$, not the time domain. Therefore, to perform a guaranteed integration on a time domain T, we need first to compute a reference on the whole time interval T. However, we believe that in the case of autonomous systems, it could be possible to use a reference computed only on a sub part of T and apply the symmetries several times with some kind of shifting window on the entirety of the time domain T. This process is illustrated in Figure 5.23.

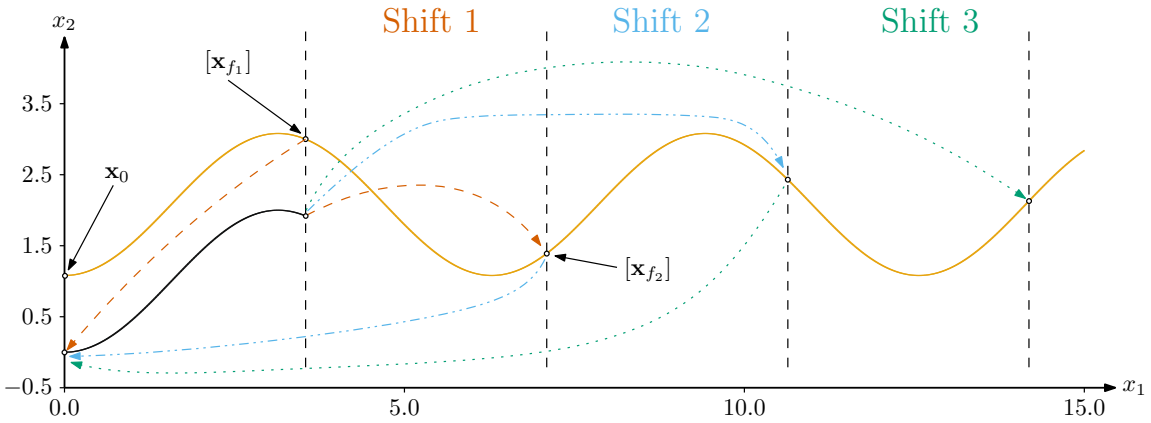


Figure 5.23: Envisioned shifting window method.

The reader should notice that the trajectory is depicted in $\mathbb{R}^2$, the time dimension is not represented. A reference, depicted in black, is computed on a short time domain ($t \in [0, 4]$) with an initial condition $\mathbf{a_0} = (0, 0)$. Using the Lie integration method, we can compute the solution for an initial condition $\mathbf{x}_0 = (0, 1)$ on the time domain $[0, 4]$ as we have done throughout this chapter. Let us denote by $[\mathbf{x}_{f_1}]$ the final box computed at $t = 4$ using the Lie integration method.

Then to compute the solution on the time domain $[4, 8]$, we use $[\mathbf{x}_{f_1}]$ as initial condition before applying the Lie integration method. This is Shift 1. The application of the symmetries is depicted by orange arrows. $[\mathbf{x}_{f_1}]$ is transported to $\mathbf{a_0}$. Moving along the reference trajectory and returning in the frame of $[\mathbf{x}_{f_1}]$, we obtain $[\mathbf{x}_{f_2}]$. This $[\mathbf{x}_{f_2}]$ will be used as initial condition for the second shift. For each step, the initial condition of the step is equal to the final condition computed at the previous one. Repeating this pattern for each shift, it is possible to integrate on the whole time domain T.

This shifting method could be useful to increase the speed of calculations when performing guaranteed integration. If one want to perform a guaranteed integration over fifty seconds, One may compute a reference using a conventional integration tool for let us say one second. Then use a shifting window to compute the result for the forty-nine following seconds.

Now, one may say that we can choose an infinitely small time domain for the reference. This would limit the pessimism on it. However there is probably a trade-off between the

pessimism induced by the Lie integration method and the one induced by the computation of the reference. Moreover, this would be coming back to a conventional step-by-step method applied to the Lie integration method. This could increase the processing time. It would be interesting to investigate this issue in the future.

5.8 Limits of the method

If the new method presented in this chapter can bring a lot of improvements in terms of processing time, enclosure sharpness and allows an easy computation of an inner approximation of the solution set, it still comes with a few limitations.

1. The first one is the need to compute the symmetries beforehand. For now, this process is not automated when in most conventional integration tools, the differentiation process to perform the integration is done in the background. One only needs to give the differential equation as input for the solver to compute an enclosure. Moreover as mentioned in Chapter 4, there may not be any (or enough) symmetries in the problem to move from any point to any other in the state space. Hence, this method is not applicable to every problem. However for most robotic systems as the one considered here the symmetries can be obvious (translations, rotations).

2. As the method needs a reference, if no analytic expression of it is available, it must be possible for a conventional tool to compute it. If no reference is available, then the method cannot be applied. This is one of the main drawbacks of the method when it comes to robotics. For instance, in the last example we used an analytical expression of the command $\mathbf{u}(t)$ to have an ODE to solve. However, most of the time, in robotics, instead of an analytical expression, we would have a tube enclosing the command in function of time. This transform the problem into a differential inclusion. Up to now, there are not so many methods to solve differential inclusions. Some work has been done using CAPD [65], but it is far from easy to use and implement, hence limiting the scope of the method. But we believe that with the recent progress in this field, this will not be a problem in the coming years.

CHAPTER 6

LIE INTEGRATION IN A ROBOTICS CONTEXT

Contents

6.1 Motivations

Let us recall the objectives that motivated this research. As stated in the beginning, we need a method to increase the precision of the prediction we could make of a robot trajectory and a better correction of its localisation without using the GNSS which is not available underwater. All this done in a guaranteed manner for safety and cost reasons. To do so, we worked on developing a new integration method which is more robust than the existing tools to the uncertainties on the initial condition. This was done because when dealing with a robotic system, its initial state is rarely exactly known, if even barely. In this chapter we will present different use cases of the method presented in the previous one. The first one could be seen as mission planning and study the feasibility of a mission. The second one is a localisation problem mixing both the Lie integration method and external measurements. But before diving straight up we will introduce briefly the concept of constraint network.

6.2 Constraint network

6.2.1 A new notation

In Section 2.3.4 we presented the notion of CSP which is one way to formalise a problem. With this, we can introduce the concept of Constraint Network (CN). Indeed, it is possible to represent our CSP under the form of a CN [83]. We will use the notation [103]. Let us bring from Section 2.3.4 the localisation problem we had. The three measurements from the different beacons gave us 3 constraints on the possible position of the robot. If we denote by $\mathbf{b}_i$ the vector (x_i, y_i) associated with the position of the buoy i, we can rewrite our problem under the form of the following CN:

$$\begin{cases} \textbf{Variables: } \mathbf{x}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, r_1, r_2, r_3 \\ \textbf{Constraints:} \\ 1.\ \sqrt{(x_1 - b_{1,1})^2 + (x_2 - b_{1,2})^2} - r_1 = 0 \quad \text{distance constraint} \\ 2.\ \sqrt{(x_1 - b_{2,1})^2 + (x_2 - b_{2,2})^2} - r_2 = 0 \quad \text{distance constraint} \\ 3.\ \sqrt{(x_1 - b_{3,1})^2 + (x_2 - b_{3,2})^2} - r_3 = 0 \quad \text{distance constraint} \\ \textbf{Domains: } [\mathbf{x}], [\mathbf{b}_1], [\mathbf{b}_2], [\mathbf{b}_3], [r_1], [r_2], [r_3] \end{cases} \tag{6.1}$$

This new notation allows us to write our problem mixing different constraints. In the example above, we have a distance constraint. We have seen in the Chapter 5 that performing a guaranteed integration could also be seen as a CSP. Thus it could be integrated as another kind of constraint in our CN as we will see in the following sections. Actually the distance constraint itself could be seen as a CN. It is possible to decompose the distance

constraint into several primitive constraints which gives us:

$$\sqrt{(x_1 - m_{1,1})^2 + (x_1 - m_{1,2})^2} - r_1 = 0 \quad \text{decomposed in} \quad \begin{cases} a = x_1 - m_{1,1} \\ b = x_2 - m_{1,2} \\ c = a^2 \\ d = b^2 \\ e = c + d \\ f = \sqrt{e} \\ g = f - r_1 \\ g = 0 \end{cases} \tag{6.2}$$

Fortunately for us, with IBEX and Codac, it is possible to write the provided formula directly when implementing, the decomposition is made in the background.

6.2.2 The contractor network

In an effort to make the computation of the solution set for problems written under the CN form easy to implement, a tool named the ContractorNetwork is available in Codac. Its implementation being out of the scope of this work, we will present it as a list of contractors to be executed, each contractor being associated with a constraint. However, the different test-cases and user cases presented in this thesis, have been used to test the capabilities and possibilities offered by such a tool throughout its development. We believe it is of great interest for the robotics community to solve a various range of problems. This is why an example of its usage is given in Example 6.1.

Example 6.1 (Contractor network implementation)**.** We will use the localisation problem to show how easy and straight forward the implementation is. We recall the different beacons position and associated range measurements, the graphical representation of the problem is given in figure 6.1.

- $\mathbf{b}_1 = (3, 4)$, $[r_1] = [3, 4]$,
- $\mathbf{b}_2 = (2, -3)$, $[r_2] = [4, 5]$,
- $\mathbf{b}_3 = (-3, 1)$, $[r_3] = [3.5, 4.5]$.

Finally we introduce a new graphical representation for the *contractor* network associated with the problem (see Figure 6.2). Each contractor (associated with a constraint) is represented under the form a of a square while domains are depicted using cables. Here we have one generic distance contractor named $\mathcal{C}_{dist}$ which is used three times and the variables $\mathbf{x}$ the position of the robot, $\mathbf{b}_i s$ position of the buoys and $\mathbf{r}_i s$ the range measurements. We define a colour scheme to distinguish the different types of variables. It will be be progressively introduce as we encounter new types of variables. In Figure 6.2, there is only one type, in light blue the variables that are not embedded in the contractor network but can be modified by it. This electric-circuit like representation will help the interested reader in the understanding of the piece of code to solve the localisation problem given in Listing 6.1.

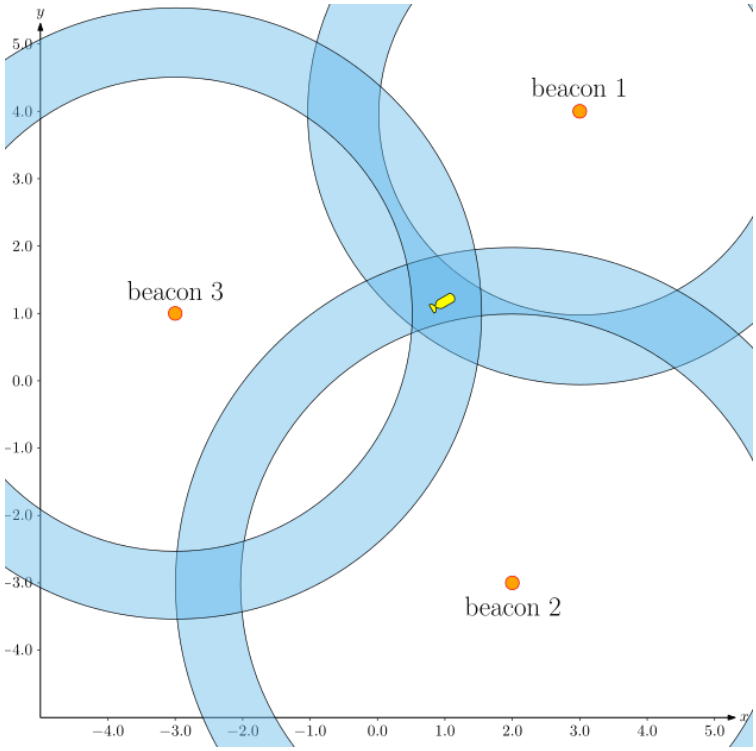


Figure 6.1: Localisation problem setup.

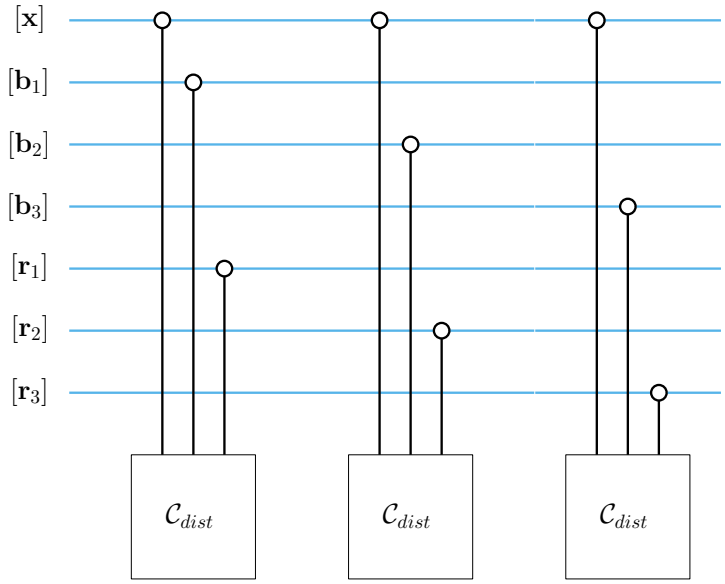


Figure 6.2: Contractor network graph.

As the reader can see in Listing 6.1, the contractor network is really simple to implement and pretty straightforward. This new tool allows us to encapsulate rather complex constraints inside one object. In order to provide a full demonstrative example, the distance function and its contractor have been redefined so that the reader can adapt the code to his/her own situation. Nevertheless there exists a predefined distance contractor object named CtcDist. Along with the distance contractor, Codac provides a few basic contractors associated with different types of measurements regularly encountered in robotics. For more examples the reader is advised to visit the tutorial section of the website dedicated to Codac.

6.2.3 Contractor on the contractor network

In Example 6.1, we solved the localisation problem for a unique search space, denoted "x" in Listing 6.1. One can notice that if we had to change the value of this search space, we would have to create a new contractor network associated with this new search space. Doing this would be very inefficient in terms of memory usage if we kept each contractor network but also in terms of computer processing time as we need to create all the new objects and connections. However changing the search space considered is a very common action. For instance, if we would like to perform the SIVIA algorithm for the localisation problem as we did in Example 2.13 we would have to change the considered search space after each bisection. To solve this issue, a small contribution of this thesis is the development of the Contractor on the Contractor Network (CtcCN). The idea came from what we have seen in Equation (6.2). The distance constraint for which we had an analytical formula could be rewritten under the form of a CN. As it has an analytical formula, Codac could easily decompose this complex constraint into primitive constraints in the background. However in our case we have three complex constraints and Codac cannot process them in one go. With the CtcCN it is possible to make one contractor on a set of complex constraints which are embedded in a ContractorNetwork object. Here is how it works:

1. We first create a basic contractor network and an abstract box.
2. Then we attach to the contractor network the different contractors applied to the abstract box. We thus have a generic contractor network
3. We generate a CtcCN using our generic contractor network, and the abstract box
4. When wanting to apply the different constraint on a specific box, the CtcCN replaces the abstract box by the considered search space.

This is all illustrated in Example 6.2.

Example 6.2 (CtcCN)**.** In this example we will show how to us the CtcCN on the localisation example to perform the SIVIA algorithm and obtain the result of Example 2.13.

The graph of the CtcCN associated with the problem is given in Figure 6.3. As one can notice, it encapsulates the previous contractor network presented. The variable $\mathbf{x}$ is now the abstract box (*depicted in pink*) that will be replaced by the value of the box $\mathbb{M}$ at each

Listing 6.1 Solving the localisation problem with the ContractorNetwork object (Codac V1)

```
1    // Initialise the search space for the robot position
2    IntervalVector x(2);
3
4    // Beacons known positions
5    IntervalVector b_1({{3},{4}});
6    IntervalVector b_2({{2},{-3}});
7    IntervalVector b_3({{-3},{1}});
8
9
10   // range measurements
11   Interval r_1(3,4);
12   Interval r_2(4,5);
13   Interval r_3(3.5,4.5);
14
15   // Create the generic distance contractor
16   Function f_dist("x[2]","m[2]","r","sqrt((x[0]-m[0])^2+(x[1]-m[1])^2)-r");
17   CtcFunction C_dist(f_dist);
18
19   // Initialising the contractor network object
20   ContractorNetwork cn;
21
22   // Adding each contractor (i.e each constraint)
23   // Equivalent to drawing wires on the graph
24   cn.add(C_dist,{x,b_1,r_1});
25   cn.add(C_dist,{x,b_2,r_2});
26   cn.add(C_dist,{x,b_3,r_3});
27
28   // Apply the constraints
29   cn.contract();
30
31   // Displaying results
32   cout <<  "Contracted search space: "<< x <<endl;
33
34   // Graphics
35   // ...
```

iteration while performing the SIVIA.
The piece of code to solve the problem is given in Listing 6.2. In this one, we used the distance contractor provided by Codac to show its usage. The attentive reader may have noticed a change in the declaration of the measurements and buoys positions.
This time the initial domains for theses variables are stored in the ContractorNetwork object. Indeed when using the contractor on the distance, the position of the robot, the distance measurement and the buoys position domains may be contracted. Hence when performing the set inversion, we may encounter cases where either of the last two may be contracted to the empty set if the search space considered is out of the solution. Thus we need to store these initial domains to reset them when considering a new search space. This type of variables will be represented as orange wires. The result obtained is displayed in Figure 6.4.

There is no inner approximation as a contractor only computes an outer approximation. To have one, we need to build the complementary contractor and use both of them to build a separator. This newly created separator used with the SIVIA algorithm will allow us to compute both the inner and outer approximation of the solution set. The code and figures are available on the web page associated with this thesis at the section static localisation.

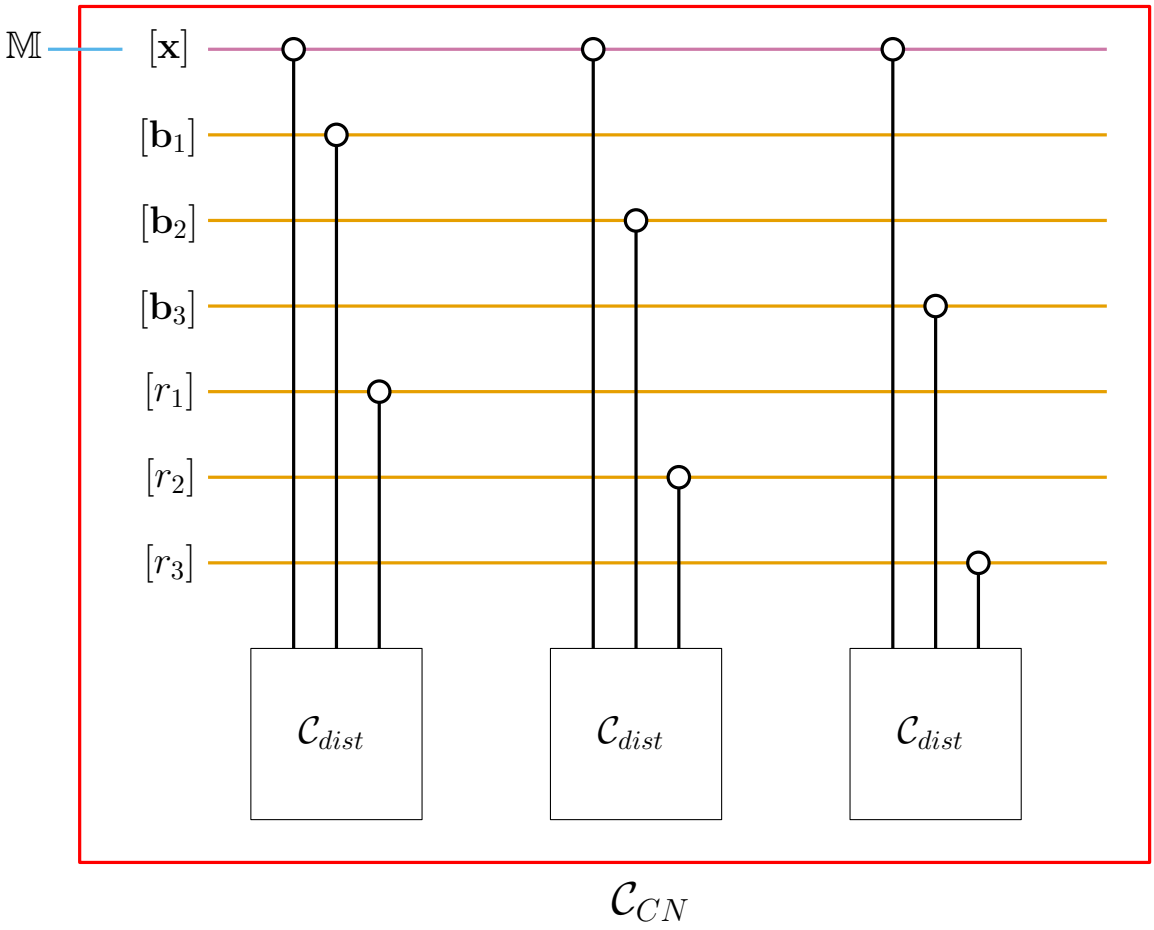


Figure 6.3: CtcCN graph for the localisation problem.

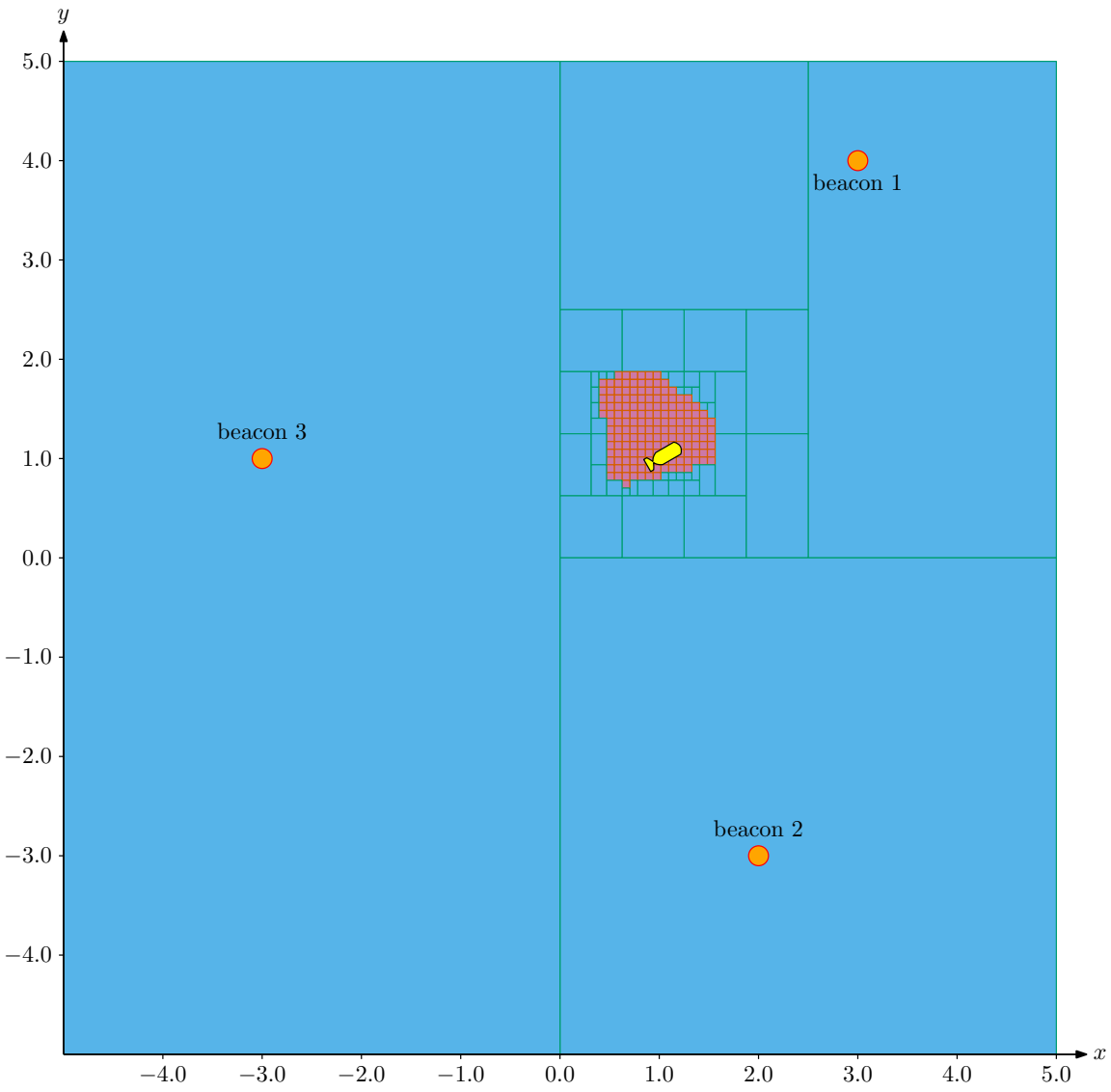


Figure 6.4: SIVIA algorithm on the localisation using the CtcCN.

Listing 6.2 Using the CtcCN on the localisation problem (Codac V1)

```
// Create the generic distance contractor
CtcDist C_dist;

/*
 *            Generating the generic localisation ContractionNetwork
 */

// Initialising the contractor network object
ContractorNetwork cn;
// Create an abstract box of the size of the search spaces we will be considering
IntervalVectorVar x(2);

/*
 * Measurements and beacons position are now stored in the ContractorNetwork
 * They will be reset to their initial value when the contract() method of
 * the CtcCN will be called before performing the contraction
 */
IntervalVector& b_1 = cn.create_interm_var(IntervalVector({{3},{4}}));
IntervalVector& b_2 = cn.create_interm_var(IntervalVector({{2},{-3}}));
IntervalVector& b_3 = cn.create_interm_var(IntervalVector({{-3},{1}}));
Interval& r_1 = cn.create_interm_var(Interval(3,4));
Interval& r_2 = cn.create_interm_var(Interval(4,5));
Interval& r_3 = cn.create_interm_var(Interval(3.5,4.5));

// Adding each contractor (i.e each constraint) using the abstract box
cn.add(C_dist,{x,b_1,r_1});
cn.add(C_dist,{x,b_2,r_2});
cn.add(C_dist,{x,b_3,r_3});

// Creating the contractor on the contractor network
CtcCN C_cn(&cn,&x);


//Initialise search space to be explored by the SIVIA
IntervalVector M({{-5,5},{-5,5}});

double epsilon = 0.1; // SIVIA accuracy
// Perform the set inversion algorithm
vector<vector<IntervalVector>> pavings = sivia(M,C_cn,epsilon);

// Graphics ...
```

6.2.4 Using a reference encapsulated in a tube

In Section 5.5.5.1, we mentioned that when using a reference obtained under the form of a tube using a conventional integration tool, we could not directly use a regular contractor or separator of Codac. This is due to the fact that for now, regular contractors cannot decompose the evaluation of a tube at a certain time as a primitive constraint. In this section we will show how this can be done using our CtcCN. Let us consider the formula we had for the flow function when working with TC3. It was:

$$\mathbf{\Phi}_t(\mathbf{x}) = \frac{\sqrt{3}\begin{pmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{pmatrix} \cdot \mathbf{a}(t)}{\sqrt{1 - \|\mathbf{a}(t)\|^2 + (4\|\mathbf{a}(t)\|^2 - 1)\|\mathbf{x}\|^2}}.$$

When performing the guaranteed integration, we were working with the constraint:

$$\mathbf{\Phi}_{-t}(\mathbf{x}) = \mathbf{x}_0 \tag{6.3}$$

Our problem was to get the value of $\mathbf{a}(t)$ which is done by evaluating the tube which encloses the reference trajectory. This complex constraint can be decomposed in a CN of two constraints:

$$\mathbf{\Phi}_{-t}(\mathbf{x}) = \mathbf{x}_0 \iff \begin{cases} \mathbf{w} = \mathbf{a}(-t) \\ \mathbf{\Phi}_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}_0 \end{cases}, \tag{6.4}$$

where

$$\mathbf{\Phi}_{\mathbf{w}}(\mathbf{x}) = \frac{\sqrt{3}\begin{pmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{pmatrix} \cdot \mathbf{w}}{\sqrt{1 - \|\mathbf{w}\|^2 + (4\|\mathbf{w}\|^2 - 1)\|\mathbf{x}\|^2}} \tag{6.5}$$

With this new formulation we can build our contractor network and the CtcCN to get the outer approximation of the solution set. We need to precise that there is one more constraint as the initial condition was under the form of a disk. The graph associated with the CtcCN named $\mathcal{C}_{CN}$ is provided in Figure 6.5. It introduces a new type of variable depicted in green. These variables are not embedded in the contractor network and will not be modified by it. It consists mainly in tubes that will be evaluated at a certain time t. In Listing 6.3, we focus only on the creation the contractor network for the outer approximation. This is another proof that multiple constraints can be gathered in the contractor network. The full piece of code used to generate Figure 5.14 or Figure 5.15 in the previous chapter is available here[1].

[1] code

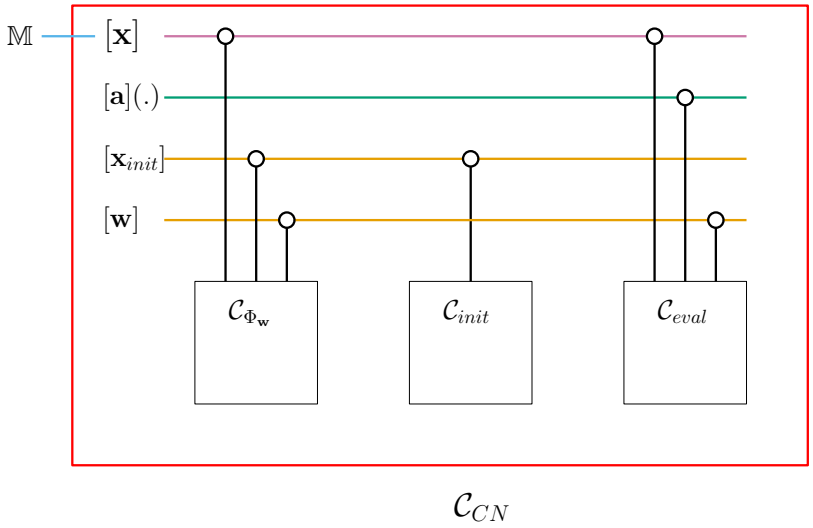


Figure 6.5: Graph of the CtcCN to perform the guaranteed integration for TC3 (Codac V1).

Listing 6.3 Contractor network evaluating the value of a tube at an instant t in TC3 (Codac V1)

```
// Create contractor for phi_w
Function phi_w("x1", "x2","t", "w1", "w2", "r1", "r2",
"((sqrt(3)*(x1*w1-x2*w2))/sqrt(1-(w1^2+w2^2)+(4*(w1^2+w2^2)-1)*(x1^2+x2^2)) - r1;\
  (sqrt(3)*(x2*w1+x1*w2))/sqrt(1-(w1^2+w2^2)+(4*(w1^2+w2^2)-1)*(x1^2+x2^2)) - r2)");
CtcFwdBwd c_phi_w(phi_w);

// Constraint on the initial condition to be a disk of radius 0.2
// centred on (1.5,1.5)
Function f_circle ("i1","i2","( (i1-1.5)^2 + (i2-1.5)^2)");
CtcFwdBwd c_init(f_circle, Interval(0, 0.04));

// Initialise evaluation of tube contractor
CtcEval c_eval;
ctc_eval.set_fast_mode(true);


ContractorNetwork cn;
IntervalVectorVar x(3); // abstract_box for x=(x_1,x_2,t)
IntervalVector& w = cn_out.create_interm_var(IntervalVector(2));
IntervalVector& x_init = cn_out.create_interm_var(IntervalVector(2));
cn.add(c_eval,{x[2],w,a}); // Constraint w = a(t)
cn.add(c_phi_w, {box,w, x_init}); // Constraint phi(-t,x)=x0
cn.add(c_initial, {x_init}); // Constraint x0 in the disk
CtcCN c_cn(&cn, &box);
```

6.3 Reachability analysis

6.3.1 Introduction

One of the field in which guaranteed integration is usually performed is reachability analysis. It is the study of the set of states a system can reach given a set of initial states and the model of its evolution through time [47, 99]. A formal definition for this is given in [2].

Definition 6.1 (Reachable set at a point in time)**.** Consider a dynamical system following a state equation of the form $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$. The set of initial states and inputs are bounded i.e $\mathbf{x}(0) \in \mathbb{X}_0, \mathbf{u} \in \mathcal{U}$. The reachable set at a certain point of time t_r is defined as the union of the possible system states at $t = t_r$:

$$\mathcal{R}(t_r) = \left\{ \mathbf{z} \in \mathbb{R} | \mathbf{z} = \int_0^{t_r} \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) dt, \mathbf{x}(0) \in \mathbb{X}_0, \mathbf{u}([0, t_r]) \in \mathcal{U} \right\} \tag{6.6}$$

where $\mathbf{u}([0, t_r]) = \bigcup_{t \in [0, t_r]} \mathbf{u}(t)$

This definition can easily be extended to the case of a time interval with Definition 6.2:

Definition 6.2 (Reachable set over a time interval)**.** The reachable set over a time interval $[t_1, t_2]$ is the union of reachable sets at points in time within the interval $t \in [t_1, t_2]$,

$$\mathcal{R}([t_1, t_2]) = \bigcup_{t \in [t_1, t_2]} \mathcal{R}(t) \tag{6.7}$$

The study of reachable sets is motivated first by safety considerations. Indeed, one may find among the set of reachable states insecure or unsafe sets which one wants to avoid. For instance, in a mobile robotics context, one may want to make sure that the robot will never reach an unsafe area or on the contrary that it will go through a mandatory location. Here is a short list of possible applications for reachability analysis [2]:

- Performance assessment of control strategies: by carrying out reachabilty analysis, one can check if the system can actually perform the mission it is supposed to perform.
- Scheduling: One may want to determine what is the best schedule in order to optimise a system. This is particularly used in the industry to increase yields of production [123]
- Controller design: In order to find the best parameters to control a system, one may use reachability analysis to ensure that the controller will not lead the system to an unsafe state.
- Deadlock: It is possible to determine if a system will not stay stuck in an unwanted certain state or set of states [98]

To carry out the different tasks presented here, numerous algorithms and methods have been developed aiming at determining either the exact reachable set in some special cases [37, 52] or an over approximation of it [4, 21].

6.3.2 Applying Lie integration method for reachability analysis

We propose here to present the use of the Lie integration method in the case of performance assessment. Staying in our mobile robotics context we will check if our system is either reaching a given area or that it will never enter a forbidden one. This reachability problem can be turned into a CSP. Indeed, setting as a constraint that our system is never inside a defined region we want to reach, if the latter is not satisfied this means that the system will reach the desired area. To illustrate this approach we consider TC4 one more time. We will also consider the initial condition of the system $\mathbf{x}_0 \in \mathbb{X}_0 = [-0.1, 0.1]^2 \times [-0.4, 0.4] \times \{0, 0\}$ and that the interval of time on which we study the system is $T = [0, 15]$. In Figure 6.6 are depicted two coloured areas. The green one, denoted $\mathcal{G}$, represents a zone we have to go through during our mission (disk centred on $(0.1, 1)^\intercal$ with radius $\sqrt{0.75}$) The red one, $\mathcal{R}$, is a forbidden area the system has to avoid (disk centred on $(1.2, 1.3)^\intercal$ with radius 0.1). In black is depicted the outer approximation of set of positions the system can have over the considered time interval T obtained in Figure 5.21 (considering only Constraint (i) in the list below).

The three constraints considered to solve our reachability problem are the following:

(i) $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \mathbf{x}(0) \in \mathbb{X}_0$

(ii) $\forall t \in [0, 15], \mathbf{x}(t) \notin \mathcal{G}$

(iii) $\exists t \in [0, 15], \mathbf{x}(t) \in \mathcal{R}$

- (i) claims that the system is initialised in the box $[\mathbf{x}_0]$ and that it satisfies the state equation,
- (ii) claims that the trajectory of the system avoids the green area $\mathcal{G}$ to be reached,
- (iii) claims that the trajectory enters the red area $\mathcal{R}$ we want to avoid at least once at some time $t_{\mathcal{R}} \in T$.

To prove that the green area $\mathcal{G}$ will be reached we have to show that (i) and (ii) cannot be both satisfied. Let us consider the CN composed of (i) and (ii).

$$\left\{\begin{array}{lll} \textbf{Variables: } \mathbf{x}(\cdot), t & & \\ \textbf{Internal variables: } \mathbf{x}_p(\cdot) & & \\ \textbf{Constraints:} & & \\ 1.\ \mathbf{x}_p(t) = \mathbf{f}(\mathbf{x}(t)) & (i) & . \\ 2.\ \mathbf{x}(0) \in [\mathbf{x}_0] & (i) & \\ 3.\ \mathbf{x}(t) \notin \mathcal{G} & (ii) & \\ \textbf{Domains: } [\mathbf{x}](\cdot), [t], [\mathbf{x}_p](\cdot) & & \end{array}\right. \tag{6.8}$$

Now, satisfying Constraint (i) is equivalent to satisfy

$$\phi_{-t}(\mathbf{x}(t)) \in [\mathbf{x}_0], \tag{6.9}$$

where $\phi_{-t}(\mathbf{x})$ is the flow function associated with the state equation of TC4 found in Section 5.6.4. Hence, it is possible to replace Constraints 1 and 2 of Equation (6.8) by Equation (6.9). This replacement is done as we have seen how to apply this constraint on a tube using a contractor network and a tube encapsulated reference obtained with CAPD in Section 6.2.4. The CN becomes

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot), t \\ \textbf{Constraints:} \\ 1.\ \phi_{-t}(\mathbf{x}(t)) \in [\mathbf{x}_0] \quad (i) \\ 2.\ \mathbf{x}(t) \notin \mathcal{G} \quad (iii) \\ \textbf{Domains: } [\mathbf{x}](\cdot), [t] \end{cases} . \tag{6.10}$$

Using the different tools presented in Section 6.2, we can apply this CN along the tube enclosing the trajectory of the system. The reader can find the circuit representation in the appendix (Figure B.1). Moreover, the piece of code used is available here[2]. As expected, the tube returned after contraction is empty. This means that both constraints cannot be satisfied at the same time, therefore, the trajectory of the system will reach the green area over the time interval considered.

In the same manner, by applying the CN formed by the constraints (i) and (iii) given below, we also obtain an empty tube as a result. We can conclude that the trajectory of the system will never cross the forbidden region.

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot), t \\ \textbf{Constraints:} \\ 1.\ \phi_{-t}(\mathbf{x}(t)) \in [\mathbf{x}_0] \quad (i) \\ 2.\ \exists t \in [t], \mathbf{x}(t) \in \mathcal{R} \quad (ii) \\ \textbf{Domains: } [\mathbf{x}](\cdot), [t] \end{cases} . \tag{6.11}$$

This example shows how the contraction-based approach to perform the guaranteed integration allows us to pair it easily with other kinds of constraints to carry out for instance a reachability analysis.

[2] code

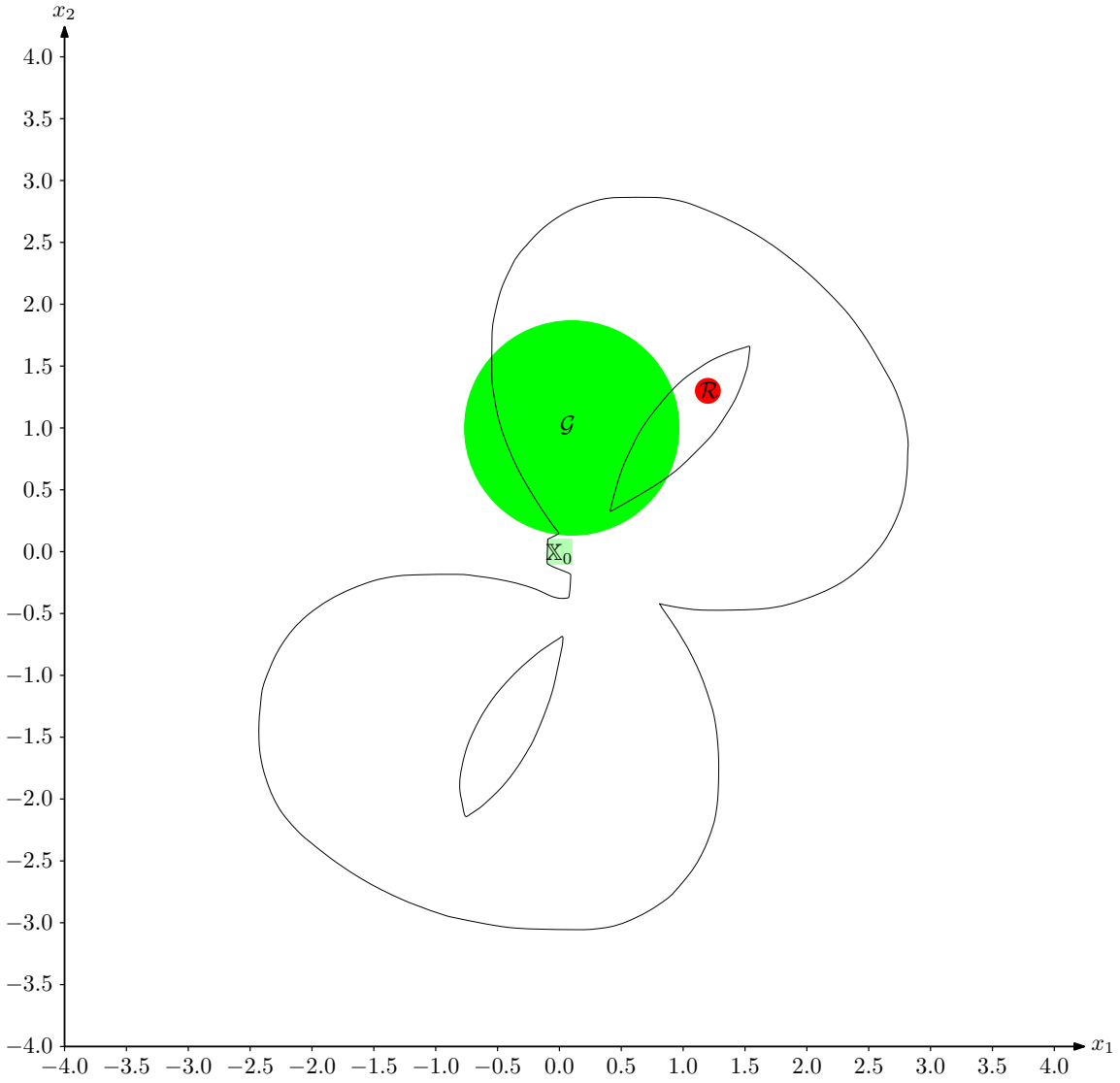


Figure 6.6: Reachability problem setup: We want to show that the green area will be reached in the considered time and that the red forbidden area will be avoided. The black shape encloses all the position the robot can have during the time interval T. It has been computed using its dynamic model (see Figure 5.21)

6.4 Localisation

Since the beginning of this work, we used the localisation example of a robot with three beacons in a static way to illustrate the different concepts presented. We considered that at a finite time all three measurements were available, which allowed us to locate the robot easily. However, in real conditions, these range measurements are often gathered in an asynchronous way. In this case, the problem we are dealing with is not static anymore and becomes a dynamic one.

6.4.1 Presentation of the problem

The problem we are going to consider is the localisation of a mobile robot which is able to communicate with a beacon when it gets close enough to it. At each time $t = k*0.5, k \in \mathbb{N}^*$ the robot tries to reach any beacon. If a response is received, a range measurement is made. The situation is presented in Figure 6.7. The true trajectory of the system is painted black. The reader will have recognised the trajectory of TC4 for which we have a state equation. An AUV is painted each time a range measurement is made. The interval for the range measurement associated with an AUV is depicted as a pair of two circles of the same colour representing the inner and upper bound.

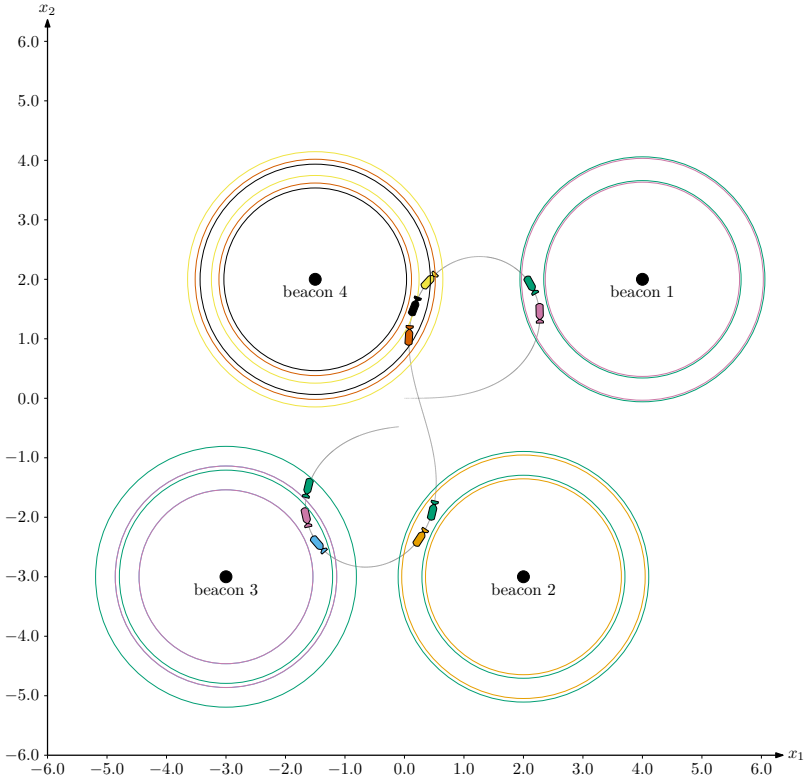


Figure 6.7: Dynamic localisation example setup

6.4.2 Localisation using range measurements only

The first, naive, idea one could have in this new situation is to use the range measurement and the time at which it has been made. Let us denote an observation by the vector

$$\mathbf{m} = \begin{pmatrix} \begin{pmatrix} b_x \\ b_y \end{pmatrix} \\ r \\ t \end{pmatrix}, \tag{6.12}$$

where $\mathbf{b}$ is the position of the beacon detected, r is the range measured and t the time associated with it. Using a mathematical object that can handle this time variable, for instance tubes, we may be able to reconstruct the trajectory of the robot using the range and the time constraint. This can be written using the CN form:

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot), \mathbf{m}_i \\ \textbf{Internal variables: } \mathbf{x}_i \\ \textbf{Constraints:} \\ 1.\ \mathbf{x_i} = \mathbf{x}(t_i) \\ 2.\ \text{dist}(\mathbf{x}_i, \mathbf{b_i}) = r_i \\ \textbf{Domains: } [\mathbf{x}](\cdot), [\mathbf{m}_i] \end{cases} \tag{6.13}$$

Using these information only, the tube enclosing the trajectory of the robot has not been contracted at all. This is due to the fact that we have no idea of the evolution of the trajectory between each measurement. Hence the robot could have moved from any point of the search space at time $t_i - dt$ to the area compliant with the range measurement r_i made at time t_i and go to any other point right after at time $t_i + dt$ where $dt \mapsto 0$. Therefore we need another information to solve our problem. This new piece of information will be the state equation of the system which is the one of TC4.

6.4.3 Adding the state equation as a constraint

As our range measurement constraint was not enough to get a sharp enclosure of our trajectory, we need to add a constraint which reflects the evolution of the vehicle through time. This dynamical constraint will be the state equation associated with the system which is the one of TC4 recalled below.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} u_1(x_4) \cdot \cos x_3 \\ u_1(x_4) \cdot \sin x_3 \\ u_2(x_4) \\ 1 \end{pmatrix}.$$

In addition we suppose that we have an enclosure of the initial state of the system denoted $[\mathbf{x}_0]$. Therefore the CN associated with this new case is the following:

$$\begin{cases} \textbf{Variables: } \mathbf{x}(\cdot), \mathbf{m}_i \\ \textbf{Internal variables: } \mathbf{x}_i \\ \textbf{Constraints:} \\ 1.\ \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \\ 2.\ \mathbf{x}(0) \in [\mathbf{x}_0] \\ 3.\ \mathbf{x_i} = \mathbf{x}(t_i) \\ 4.\ \text{dist}(\mathbf{x}_i, \mathbf{b_i}) \in [r_i] \\ \textbf{Domains: } [\mathbf{x}](\cdot), [\mathbf{m}_i] \end{cases} \tag{6.14}$$

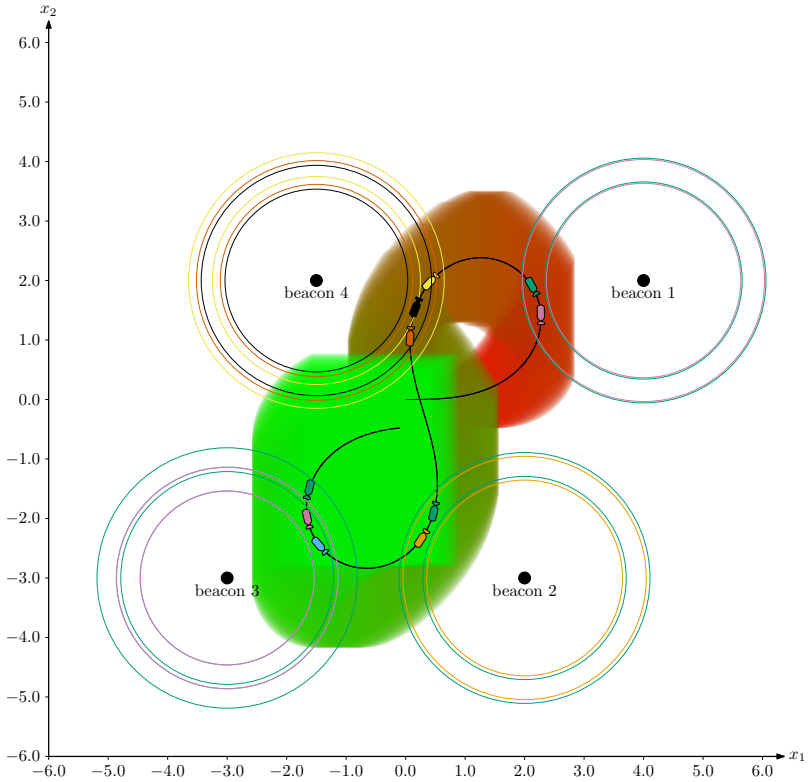
The issue of using Constraint 1 has been addressed throughout this work using different integration methods. Using this CN to implement its contractor network counterpart, we obtain the results depicted in Figure 6.8a and Figure 6.8b. The enclosures go from red (beginning of the trajectory) to green (end of the trajectory). As the reader has probably noticed, the enclosures obtained are sharper than the ones in Section 5.6.5.2 when we were comparing the tubes resulting from integration only. As expected the result obtained with the Lie integration method is far better than the one achieved using CAPD. The enclosure of the trajectory using only the dynamical constraint was already sharper in Section 5.6.5.2. However there is one reason that makes Lie integration method even more interesting in this constraint-based scheme. It allows us to propagate constraints both forward **and** backward in time when the other methods can only propagate them forward. Thus measurements done during the mission can be used to get a better enclosure of the initial condition using constraint propagation backward in time. Then using this new and more precise initial condition we can integrate forward in time and get a better enclosure of the trajectory. We will explore this last property in the upcoming section.

6.4.4 Advantage of the Lie method

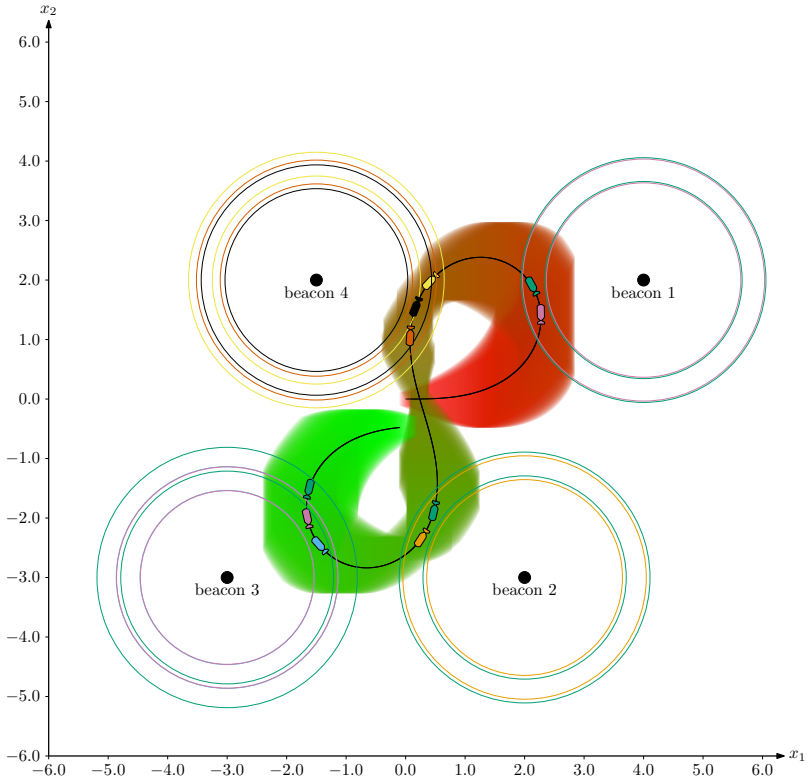
6.4.4.1 Estimating the initial condition

In the previous section, we supposed that we had a mildly uncertain initial condition. This allowed us to use conventional guaranteed integration tools in order to make a prediction on the robot's trajectory which is then corrected by the range measurements made during the mission in order to get an as sharp as possible enclosure of the real trajectory. However in this case the calculation of the prediction does not benefit from the range measurements made as we cannot propagate the constraint backward. We will show that with the Lie integration method it is possible, using only range measurements and the robot's dynamics, to estimate what was its initial state and use it to determine its location. Let us introduce some notations we will use throughout this section.

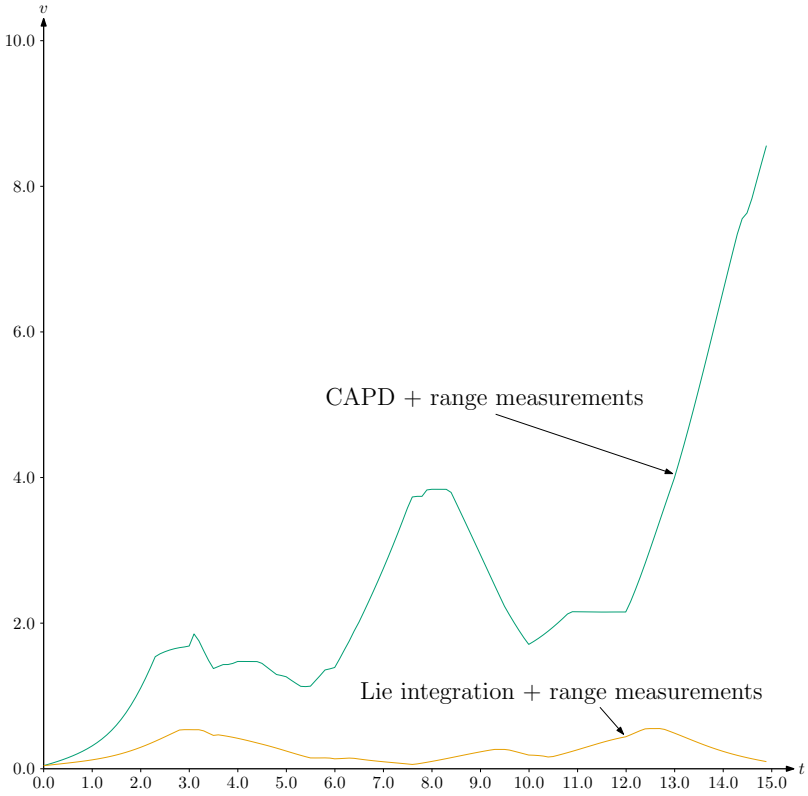
- First the distance function g which computes the range between the robot and the beacon detected at time t_i.
- Then $\mathbf{\Phi}_t$ the flow function of the system.



(a) Localisation using the trajectory predicted by CAPD corrected with range measurements



(b) Localisation using the Lie integration method coupled with range measurements



(c) Evolution of the volume of tube enclosing the robot trajectory using prediction from dynamic model and range measurements

- r_i the distance measured at time t_i between the vehicle and the beacon detected.
- $\mathbb{Z}_i$, the set of states which are compliant with the measurement r_i, $\mathbb{Z}_i = \{\mathbf{z}_i \in \mathbb{R}^3 | \mathbf{g}_i(\mathbf{z}_i) = r_i\}$
- $\mathbb{Q}_i$ the set of possible initial conditions from which it is possible to reach $\mathbb{Z}_i$. We have $\mathbb{Q}_i = \{\mathbf{q}_i \in \mathbb{R}^3 | \Phi_{t_i}(\mathbf{q}_i) \in \mathbb{Z}_i\} \iff \mathbb{Z}_i = \mathbf{\Phi}_{t_i}(\mathbb{Q}_i)$ but also $\mathbb{Z}_i = \{\mathbf{z}_i \in \mathbb{R}^3 | \Phi_{-t_i}(\mathbf{z}_i) \in \mathbb{Q}_i\} \iff \mathbb{Q}_i = \mathbf{\Phi}_{-t_i}(\mathbb{Z}_i)$.
- $\mathbb{Q}$ the estimation of the initial condition which is compatible with all measurements performed: $\mathbb{Q} = \bigcap_i \mathbb{Q}_i$.
- $\mathbb{X}_t$, the estimation of the state of the robot at time t, $\mathbb{X}_t = \{\mathbf{x} \in \mathbb{R}^3 | \Phi_{-t}(\mathbf{x}) \in \mathbb{Q}\}$, we also have $\mathbb{Q} = \{\mathbf{q} \in \mathbb{R}^3 | \Phi_t(\mathbf{q}) \in \mathbb{X}_t\}$.

To help the reader understand how the approximation of the initial state $\mathbb{Q}$ is computed, we will proceed step by step. Each new step starts when a new piece of information is available to us. Let us summarise the different pieces of information we have at time $t = 0$. We know that the state of the robot could be anywhere in the state space. The dynamics of the system is known. If we let the system evolve, until the first measurement at time t_1 depicted as the pink AUV in the top left corner in Figure 6.7, we do not gather any new information we could use to locate the system.

Now at time t_1 a new range measurement r_1 is available. With it, we can compute an approximation of the position of the robot $\mathbb{Z}_{t_1}$ as $z_1 = g(r_1)$. This approximation projected on the first two dimensions (x_1, x_2) is depicted in Figure 6.10. We recall that the inner approximation is composed of the white part and the outer approximation gathers the white and pink parts. For this figure and the following, the interval range measurement is depicted by two black circles (lower and upper bounds). The position of the robot when the measurement is performed is painted black. Finally the true trajectory of the system is painted black. As expected, the approximation we get is a ring as we only have a range measurement. With this approximation of $\mathbb{Z}_1$, it is then possible to compute an enclosure for $\mathbb{Q}_1$ as we know the vehicle dynamics. We recall that $\mathbb{Q}_1$ is the set of all possible initial states for which the measurement r_1 could have been made after the system evolution between 0 and t_1. This second approximation is presented in Figure 6.11. The true initial state is represented by the yellow AUV. Using $\mathbb{Q}_1$, it is possible to compute the estimation of the initial state, $\mathbb{Q}$. As we only have one measurement for now, $\mathbb{Q}_1 = \mathbb{Q}$. This full process can be summarised by the following CN with $i \in \{1\}$:

$$\begin{cases} \textbf{Variables: } r_i, \mathbf{q} & \\ \textbf{Internal variables: } \mathbf{z}_i, \mathbf{q}_i, & \\ \textbf{Constraints:} & \\ 1.\ \mathbf{z}_i = g(r_i) & \forall i \\ 2.\ \mathbf{q}_i = \mathbf{\Phi}_{-t_i}(\mathbf{z}_i) & \forall i \\ 3.\ \mathbf{z}_i = \mathbf{\Phi}_{t_i}(\mathbf{q}_i) & \forall i \\ 4.\ \mathbf{q} = \mathbf{q}_i & \forall i \\ \textbf{Domains: } [\mathbf{q}], [r_i], [\mathbf{q}_i], [\mathbf{z}_i] & \end{cases} \tag{6.15}$$

This CN stays valid in case of multiple measurements. As we have done in the reachability example we can create a CtcCN on this CN that we will denote $\mathcal{C}_\mathbf{q}$. Its circuit representation is given in Figure 6.9 We have used all of our pieces of information available for now, we can wait until the next measurement.

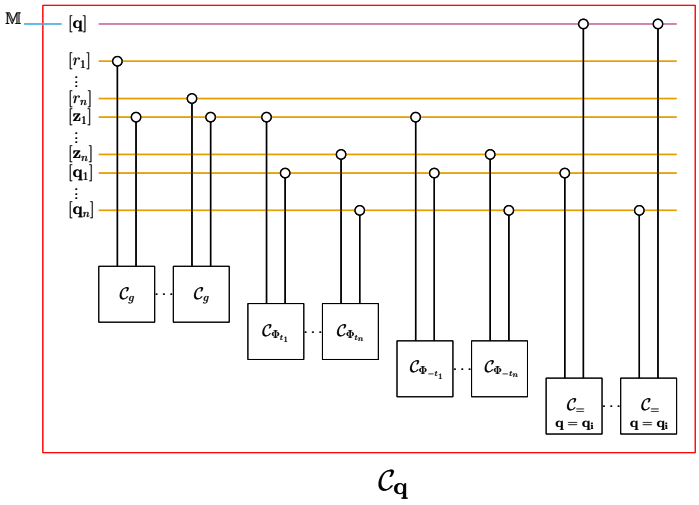


Figure 6.9: Circuit representation of $\mathcal{C}_\mathbf{q}$

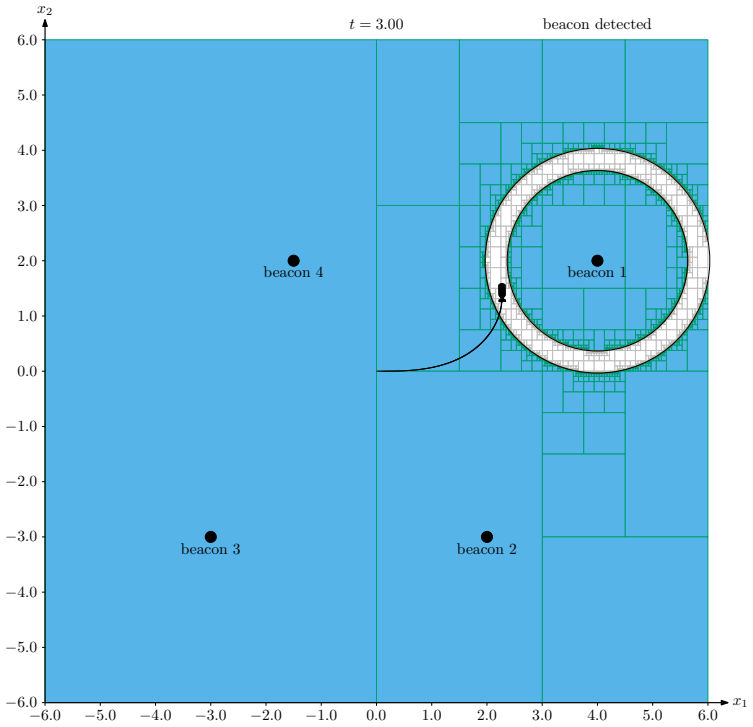


Figure 6.10: Robot position approximation $\mathbb{Z}_1$ at time t_1.

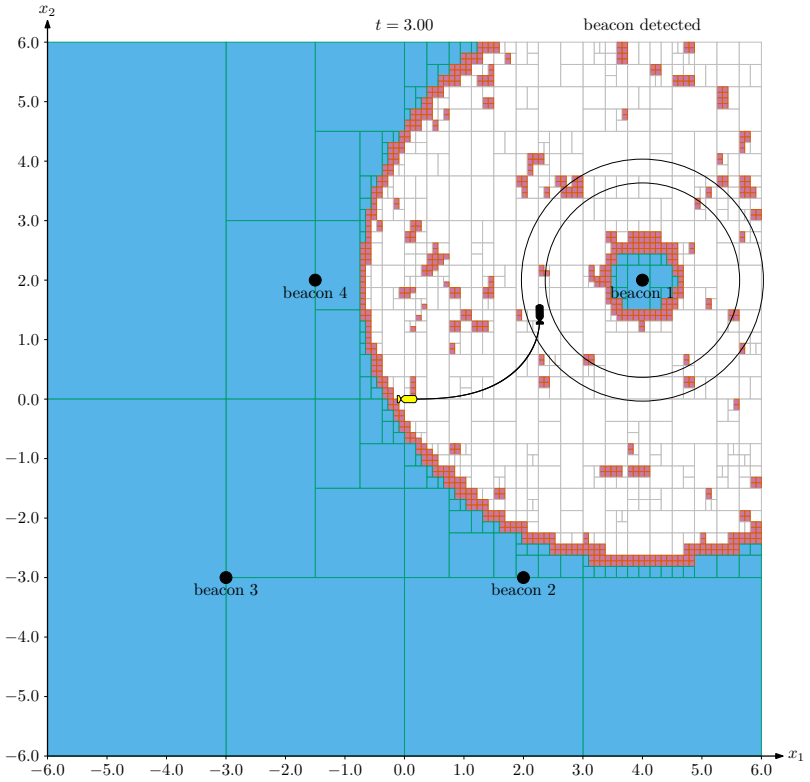


Figure 6.11: Robot initial position approximation $\mathbb{Q}_1$ at time t_1.

The next measurement at time t_2 allows us to compute the approximations for both sets $\mathbb{Z}_2$ and $\mathbb{Q}_2$. With the latter, we get a new constraint on $\mathbb{Q}$ as it is the intersection of the $\mathbb{Q}_i s$. As one can see on Figure 6.13, the enclosure does not get much better. This is due to the fact that we are detecting the same beacon. Hence the system can still come from anywhere around the beacon.

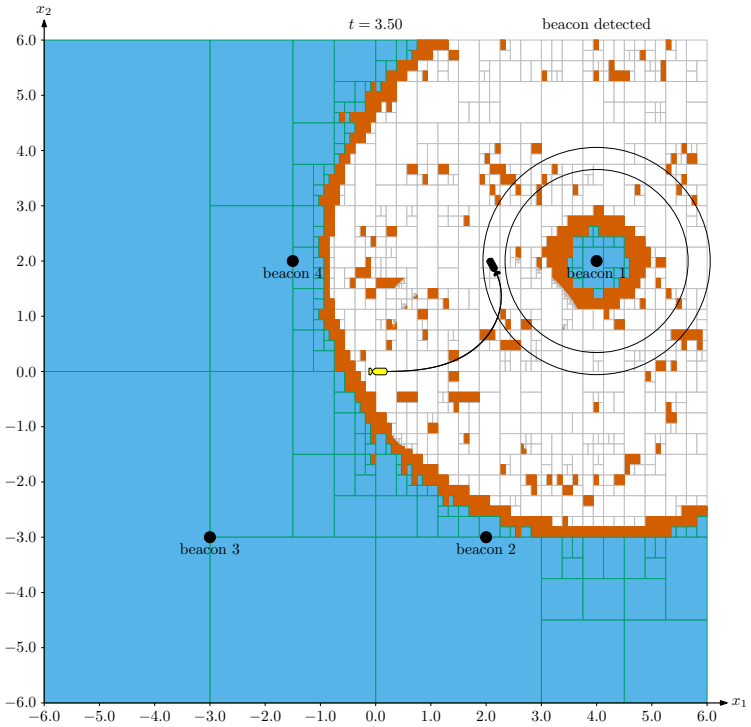


Figure 6.12: Robot initial position approximation $\mathbb{Q}_2$ at time t_2.

At time t_3 we detect a new beacon. As done with the two previous measurements, we compute both approximations for $\mathbb{Z}_3$ and $\mathbb{Q}_3$. We only represent the approximation of the set $\mathbb{Q}_3$ in Figure 6.14. With this last figure, one can already feel that the sharpness of the approximation for $\mathbb{Q}$ will be greatly increased with this new measurement. The new estimation for $\mathbb{Q}$ is given in Figure 6.15.

Repeating the same operations for each measurements, we obtain at time $t = 15$ the approximation for $\mathbb{Q}$ displayed in Figure 6.16. We get a rather precise estimation compared to the mildly uncertain initial box we used in the previous version of the problem. This shows the true power of the Lie method. This ability to propagate constraints both backward and forward allows us to estimate the initial state which, we recall, was not known at the beginning of the problem.

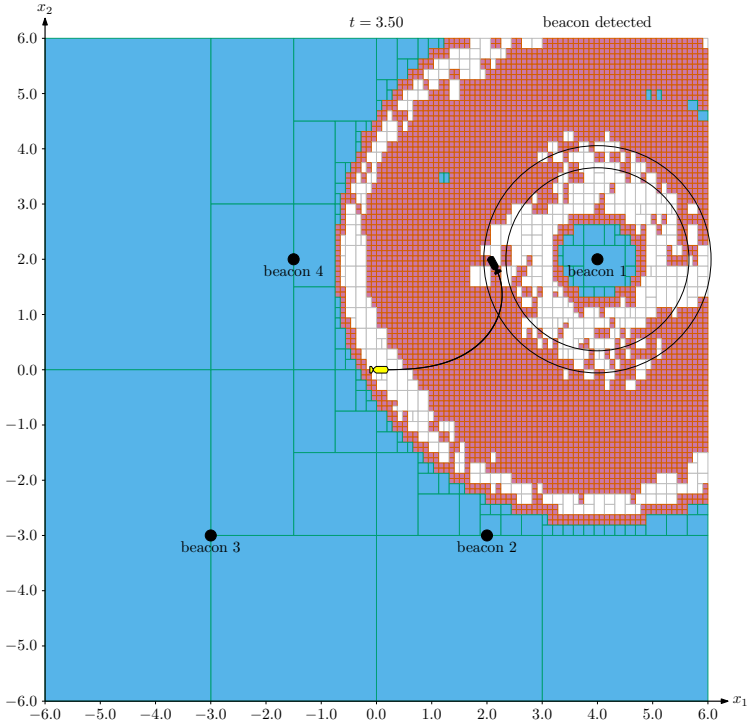


Figure 6.13: Robot initial position approximation $\mathbb{Q}$ after 2 measurements.

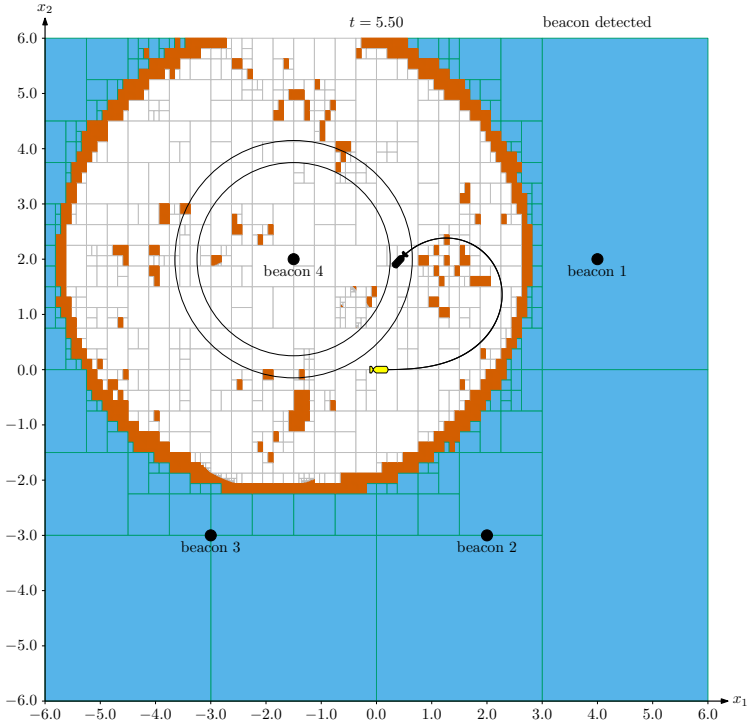


Figure 6.14: Robot initial position approximation $\mathbb{Q}_3$ at time t_3.

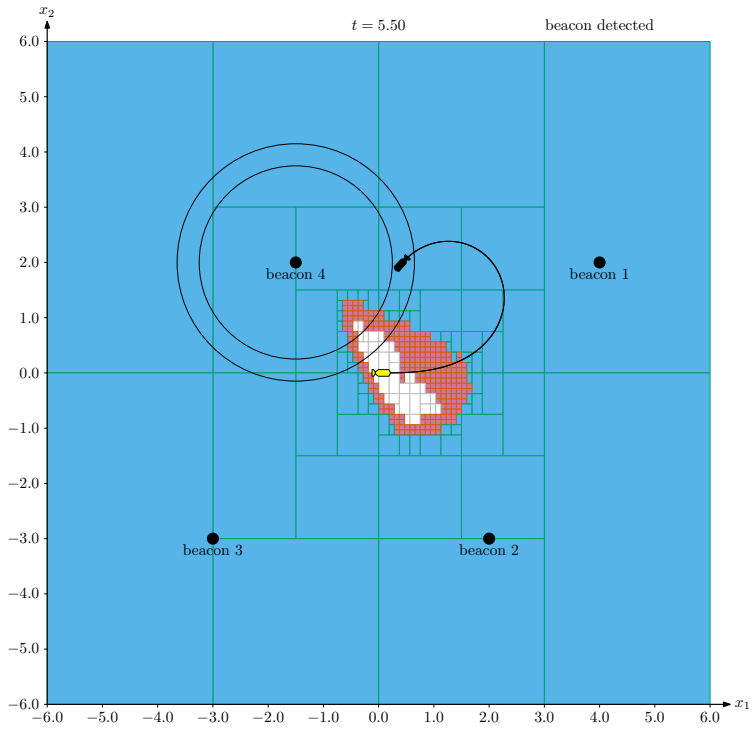


Figure 6.15: Robot initial position approximation $\mathbb{Q}$ after 3 measurements.

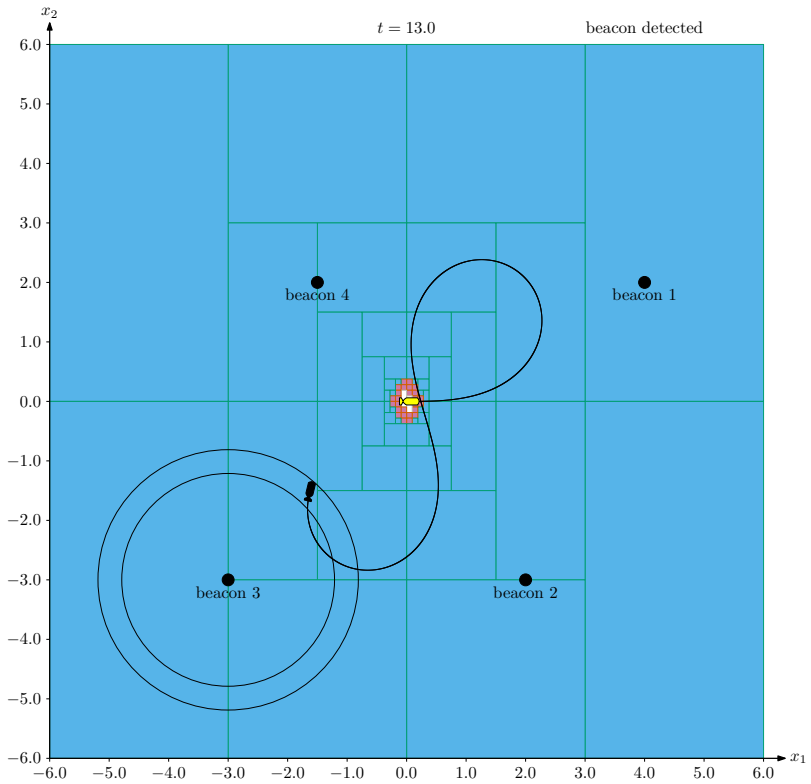


Figure 6.16: Robot initial position approximation $\mathbb{Q}$ after 9 measurements.

6.4.4.2 Using our initial condition estimation to solve the localisation problem

The capability to estimate the initial state of the vehicle is of great interest in our localisation problem with an unknown initial state. Indeed, if we were to estimate the robot's state over time using conventional integration techniques, having an entirely unknown initial condition would have us unable to solve the problem. As we have seen in Chapter 3, the bloating effect would appear quickly and we would not be able perform the integration. With this new estimation, we can now propagate the dynamics constraint forward to get an approximation of the system at time t. Using the fact that

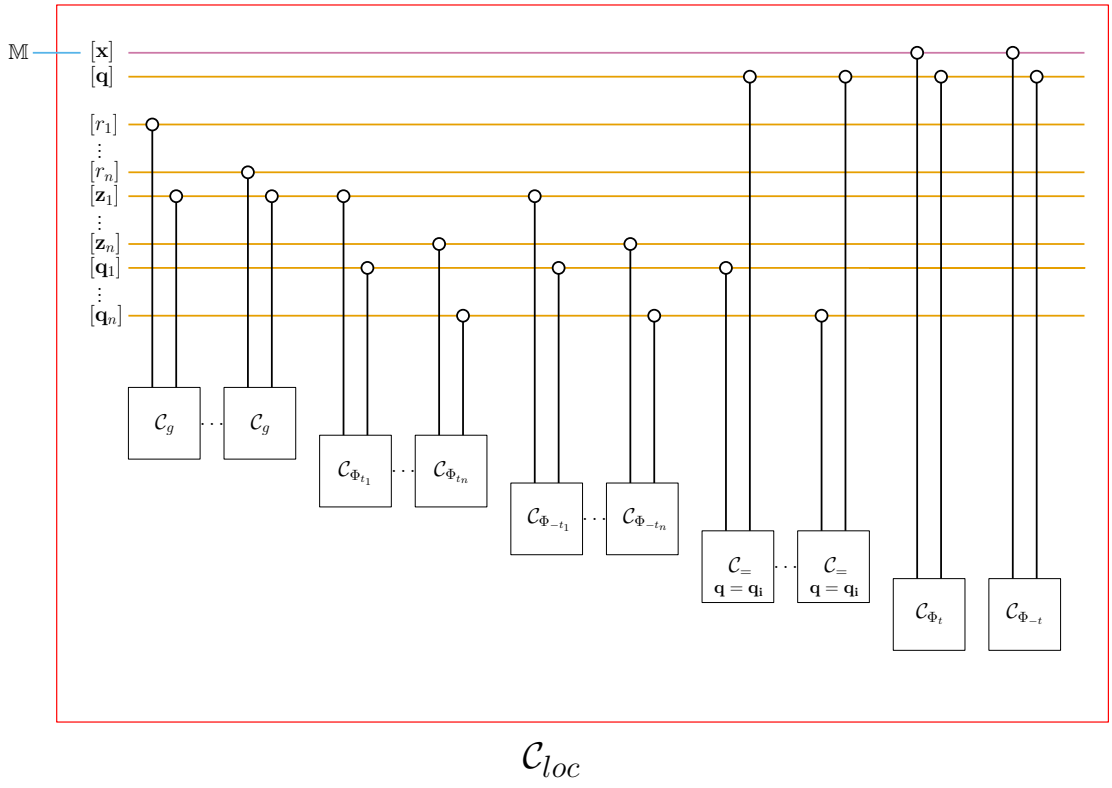
$$\begin{cases} \mathbb{X}_t = \mathbf{\Phi}_t(\mathbb{Q}), \\ \mathbb{Q} = \mathbf{\Phi}_{-t}(\mathbb{X}_t), \end{cases}, \tag{6.16}$$

we can enhance the CN presented in Equation (6.15) in order to compute the approximation of $\mathbb{X}_t$. It becomes:

$$\begin{cases} \textbf{Variables: } r_i, \mathbf{q}, \mathbf{x} \\ \textbf{Internal variables: } \mathbf{z}_i, \mathbf{q}_i, \\ \textbf{Constraints:} \\ 1.\ \mathbf{z}_i = \mathbf{g}(r_i) & \forall i \\ 2.\ \mathbf{q}_i = \mathbf{\Phi}_{-t_i}(\mathbf{z}_i) & \forall i \\ 3.\ \mathbf{z}_i = \mathbf{\Phi}_{t_i}(\mathbf{q}_i) & \forall i \\ 4.\ \mathbf{q} = \mathbf{q}_i & \forall i \\ 5.\ \mathbf{q} = \mathbf{\Phi}_{-t}(\mathbf{x}) \\ 6.\ \mathbf{x} = \mathbf{\Phi}_t(\mathbf{q}) \\ \textbf{Domains: } [r_i], [\mathbf{q}], [\mathbf{x}], [\mathbf{q}_i], [\mathbf{z}_i] \end{cases}. \tag{6.17}$$

The circuit representation of the CtcCN, $\mathcal{C}_{loc}$ associated can be found in Figure 6.17. One can notice that it is an extension of the graph in Figure 6.9, where we added the relationships between $\mathbf{x}$ and $\mathbf{q}$ Of course for a defined time $t \in [0, t_f]$ the only range measurements considered to compute the approximation are the r_is made at time $t_i < t$. Hence the more range measurements we have, the better the approximation of $\mathbb{X}_t$ gets as the sharpness of the enclosure for $\mathbb{Q}$ increases as seen in the previous section. The reader may be confused by the fact that at each time t_i when a measurement is made, both $\mathbb{Z}_i$ and $\mathbb{X}_{t_i}$ are sets gathering the possible states of the vehicle at time t_i. The difference between these two sets is that for all $i > 1$, $\mathbb{X}_{t_i}$ takes into account all the previous measurements, when $\mathbb{Z}_i$ does not. This is due to the fact that $\mathbb{Q}$ "stores" the information given by each new measurement as it is the intersection of all initial state approximation made thanks to each measurement. Some results using this method are displayed below. One can notice that the approximation gets better as the number of range measurements increases. It is even better when the beacon detected is not the same between two consecutive measurements. As seen in Chapter 5, it is possible to compute both discrete and continuous sets. A video showing the evolution of the approximation of $\mathbb{X}_t$ is available here[3]. The upper part shows the evolution of the enclosure of $\mathbb{X}_t$ when the lower part depicts the evolution of the approximation of $\mathbb{Q}$.

[3] https://youtu.be/AC2HxxmgaE8



$\mathcal{C}_{loc}$

Figure 6.17: Circuit representation of $\mathcal{C}_{loc}$

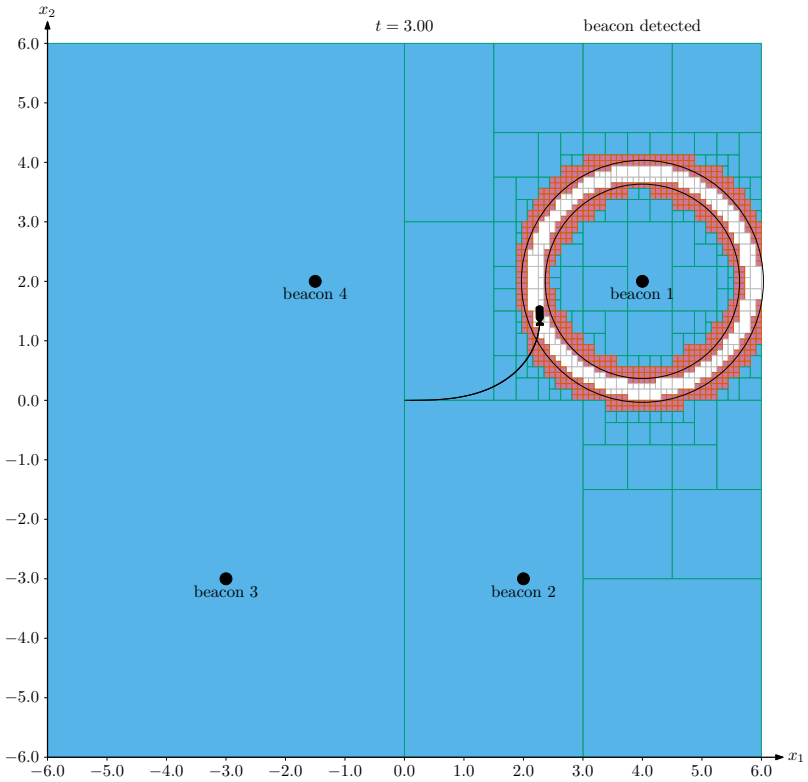


Figure 6.18: $\mathbb{X}_{t_1}$

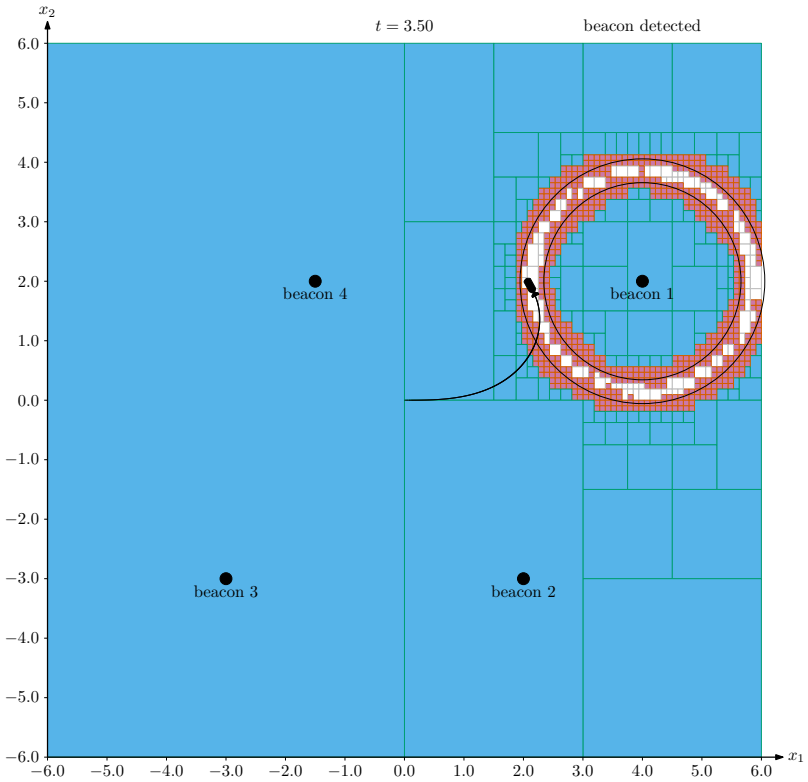


Figure 6.19: $\mathbb{X}_{t_2}$

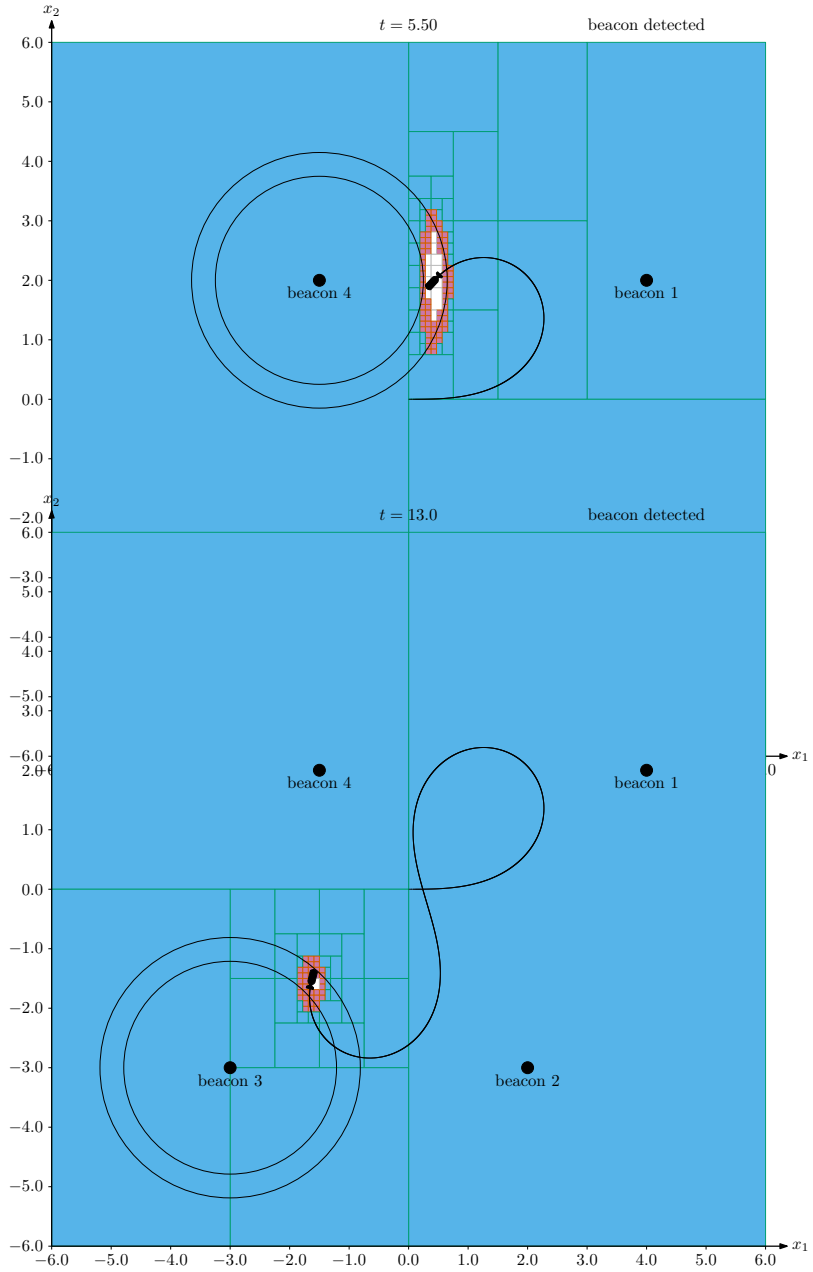


Figure 6.21: $\mathbb{X}_{t_9}$

CHAPTER 7

CONCLUSION AND ENVISIONED FUTURE WORK

Contents

7.1 Conclusion

This research has been motivated by the need to improve the localisation of low-cost underwater robots in the context of offshore infrastructures monitoring. Being deprived of high-end sensors, the robot has to navigate relying on dead-reckoning mainly. However, as seen in the introduction, the results returned by the INS alone are not sufficient to perform long missions. Especially when it is a low-cost one. Therefore we aimed at developing a new method that would enhance the results, based on the extraction of symmetry properties from the dynamic model of the system.

In a first part, Chapter 2, Chapter 3 and Chapter 4 present the different mathematical concepts used throughout this thesis.

After introducing the field of dynamical systems and their link with differential equations, the first one focuses on the set-membership approach we chose to handle our variables. The basics of interval analysis and the tubes to deal with dynamical problems are introduced.

Then Chapter 3 addresses the problem of guaranteed integration. As we have to ensure the safety of the system we need to guarantee our results when performing computations. The different existing methods are presented in this chapter. We demonstrate their functioning processes and limits induced by them when it comes to handling the large uncertainties occurring in a robotic context.

Finally, Chapter 4 introduces the concept of Lie Groups and especially Lie symmetries. The process to find such symmetries is explained and the new notion of transport function is detailed. We have shown how Lie symmetries can applied to differential equations to find solutions for different initial conditions from a previously computed one.

Then in a second part, the new Lie Integration method associating the different tools presented in the previous chapters is explained in Chapter 5. We also demonstrated how the guaranteed integration can be seen as a set inversion problem. We applied our new method onto four different test cases and compared the results with already existing tools to assess its efficiency when dealing with uncertainties on the initial condition.

The last piece of this work demonstrates the efficiency of the Lie integration method by solving first a reachability problem and then the localisation problem defined in the introduction that motivated this research.

7.2 Contributions

The main contributions of this thesis are presented below.

First the notion of "transport function" that allows to find solutions of differential equations for any initial condition in the state space using a known reference. This notion is essential in the development of a new integration method.

Using the previously introduced tool we developed a new integration method that is able to handle large uncertainties provided the system has enough symmetries. We demonstrated

its efficiency in both robustness and computer processing time. Moreover, while we applied this method in a set-membership context, it is not limited to it. Thus other works might benefit from it. These two contributions were initially introduced at SWIM 2019[26] and are the subject of a journal paper entitled "Lie symmetries applied to interval integration" provisionally accepted in Automatica [27].

This work has also contributed to the development of the Codac library. It added the ability to easily use CAPD tools which are the state of the art in terms of guaranteed integration. This allows an easier use of contractor tools with the results returned by CAPD which helped us in the resolution of the localisation problem. In addition, the contractor on the ContractorNetwork allows the use of complex sets of constraints on particular objects in Codac.

7.3 Prospects

Solve differential inclusions. As mentioned at the end of Chapter 5, it is only possible for now to solve analytical differential equations with the Lie method integration method. Therefore a compromise has to be done when dealing with real applications. We have seen with the fourth test case that, for now, we need to have an analytic expression of the command of the robot over time. However in the future being able to generate sharp enclosure for the reference using the real command sent during the mission could help improving the results in the post-processing of the operation.

Handling both space and time displacement. In the examples presented throughout this work we demonstrated how symmetries could be used to compute solutions using any point of the state space as initial condition using a reference. However, as stated in the conclusion of Chapter 5 nothing prevents us from using a reference with a short time domain and use it as a sliding window to compute the solution of the integration on large time domain. This could be useful when computing the reference is computationally expensive. It is a subject to be explored.

Apply the symmetries with other filters. We used the symmetries in the context of guaranteed integration to increase the speed of calculation but also to compute both an inner and an outer approximation of the solution set easily. However, numerical integration is rather common in numerous fields. As we quickly mentioned it in the comparisons section of Chapter 5, it could be very efficient in the case of particle filters where multiple simulations need to be performed. As it is not computationally heavy, the speed of simulations could be greatly increased but it may also be usable with less high-end hardware. This means that it may be used in embedded systems in the future.

Compute the transport function automatically. In Chapter 4, we have shown how to find the expression of the transport function. Up to now this step still has to be performed by hand before getting to the implementation. One of the key points to address for this method to be easier to use and thus expand, is the development of a tool able to generate the transport function using a reference provided as input and the state equation. This last prospect is probably the most ambitious one but the one that most people working with the contributions of this thesis would benefit from.

APPENDIX A

ADDITIONS TO CHAPTER 5: A NEW GUARANTEED INTEGRATION METHOD

A.1 Figure: Integration of TC3 using the analytic expression of the reference

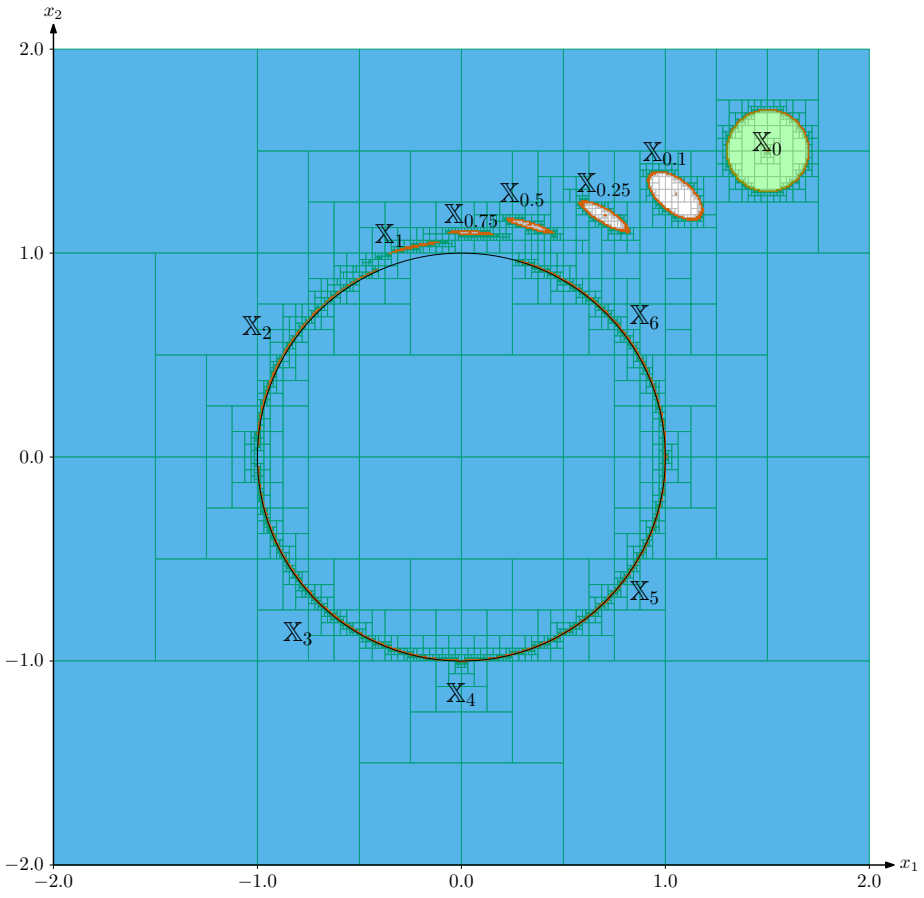


Figure A.1: Integration of multiple discrete sets for TC3, using the analytic expression to compute the reference

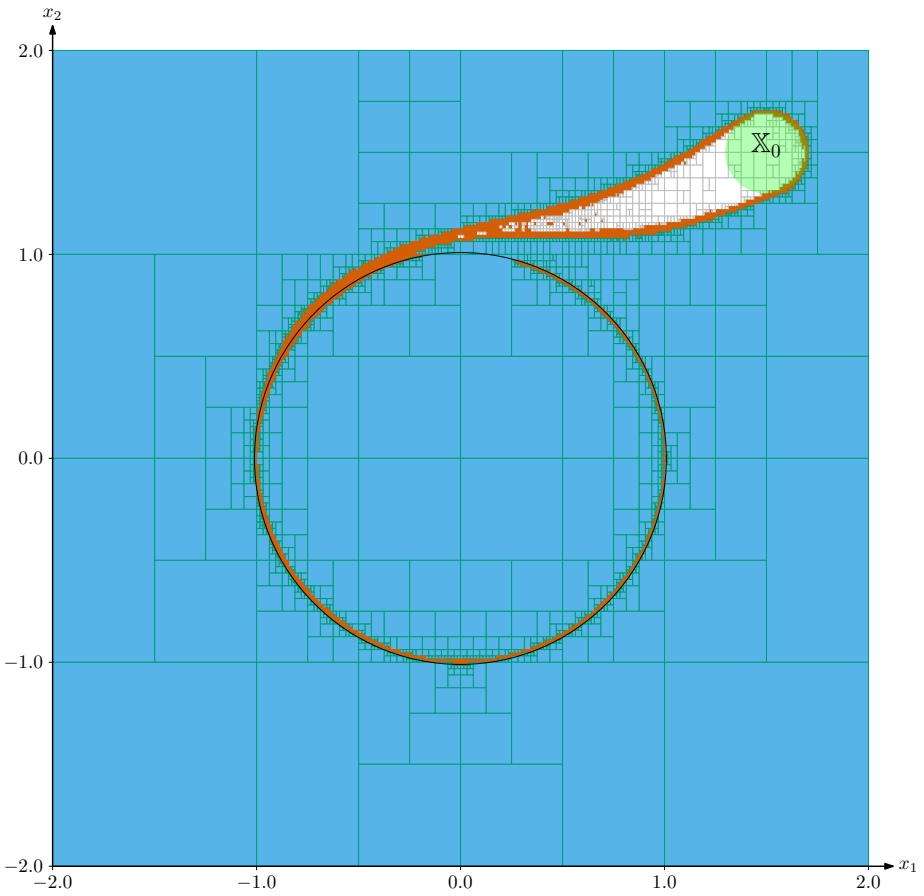


Figure A.2: Continuous integration applied on TC3 using analytic expression to compute the reference

A.2 Lawnmower pattern error evolution

Consider the system of TC4:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} u_1(t) \cdot \cos x_3 \\ u_1(t) \cdot \sin x_3 \\ u_2(t) \end{pmatrix}$$

The first two coordinates at time T from the last know position at time t_k can be written as

$$x_1(T) = \int_{t_k}^{T} u_1(t) \cos(x_3(t)) dt x_2(T) = \int_{t_k}^{T} u_1(t) \sin(x_3(t)) dt$$

The estimated state $\hat{\mathbf{x}}$ when considering a scale factor k close to 1 from the DVL and a constant bias b close to 0 on the angle measurement is:

$$\hat{x}_1(T) = \int_{t_k}^{T} k u_1(t) \cos(x_3(t) + b) dt$$
$$\hat{x}_2(T) = \int_{t_k}^{T} k u_1(t) \sin(x_3(t) + b) dt$$

Using trigonometry we obtain:

$$\begin{aligned} \hat{x}_1(T) &= k \cos(b) x_1(t) - k \sin(b) x_2(t) \\ \hat{x}_2(T) &= k \sin(b) x_1(t) + k \cos(b) x_2(t) \end{aligned} \tag{A.1}$$

Then the navigation (or position) error is :

$$e(t) = \sqrt{(\hat{x_1}(t) - x_1(t))^2 + (\hat{x_2}(t) - x_2(t))^2}.$$

Thus,

$$e^2(t) = [k \cos(b) x_1(t) - k \sin(b) x_2(t) - x_1(t)]^2 + [k \sin(b) x_1(t) + k \cos(b) x_2(t) - x_2(t)]^2,$$

which is equivalent to

$$e^2(t) = [x_1^2(t) + x_2^2(t)][k^2 + 1 - 2k \cos(b)].$$

We denote by $d(t)$, the distance from the last known point at time t_k, then:

$$e(t) = d(t)[k^2 + 1 - 2k \cos(b)]. \tag{A.2}$$

We simulated the system obtaining the "true" trajectory in blue in Figure A.3. We then proceeded to compute the estimate using Equation (A.1). As one can see the error is increases when we get further from the starting point and decreases when we get close to it.

We can then compare the error point to point from the true trajectory and the one estimated, and the error computed using the formula given by Equation (A.2). In Figure A.4, one can see that both errors are the same and evolves linearly with the distance from the last known point.

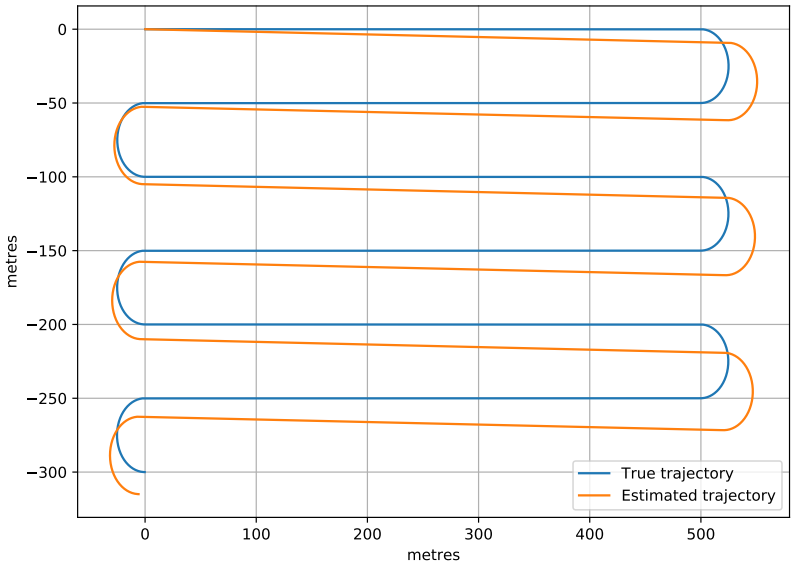


Figure A.3: A simulated trajectory of the system (in blue) and the estimate computed using Equation (A.1) in orange

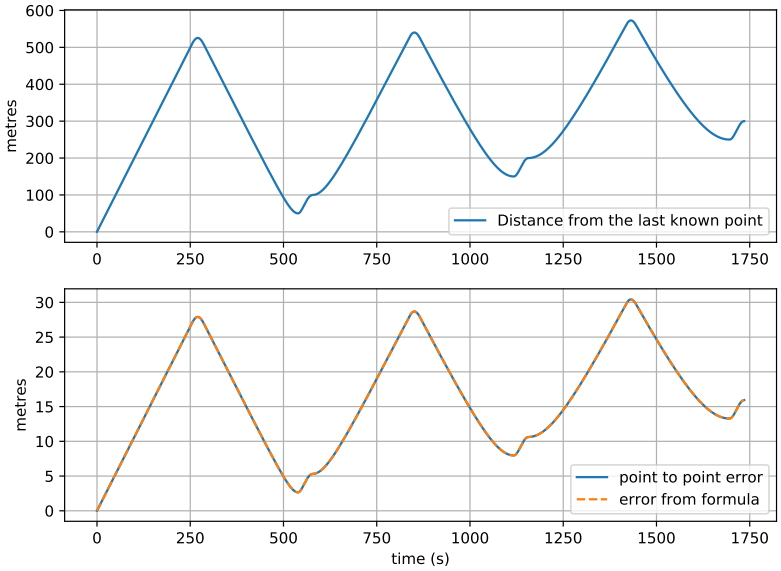


Figure A.4: Evolution of the error estimation in function of the distance form the last known point

APPENDIX B

ADDITIONS TO CHAPTER 6: LIE INTEGRATION IN A ROBOTICS CONTEXT

B.1 Circuit representation for the reachability example

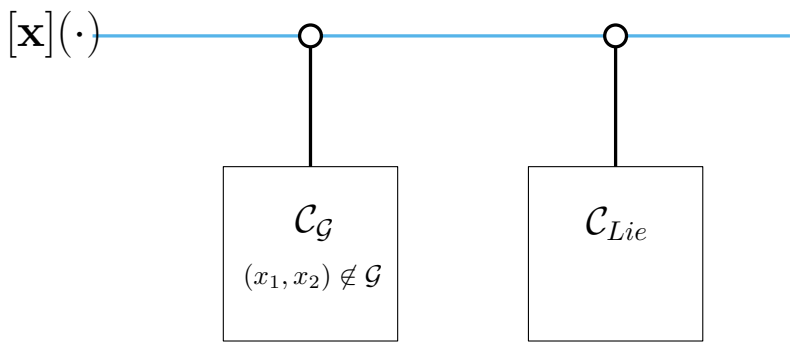


Figure B.1: Circuit representation of the constraint network that checks if an area will be reached given an uncertain initial condition

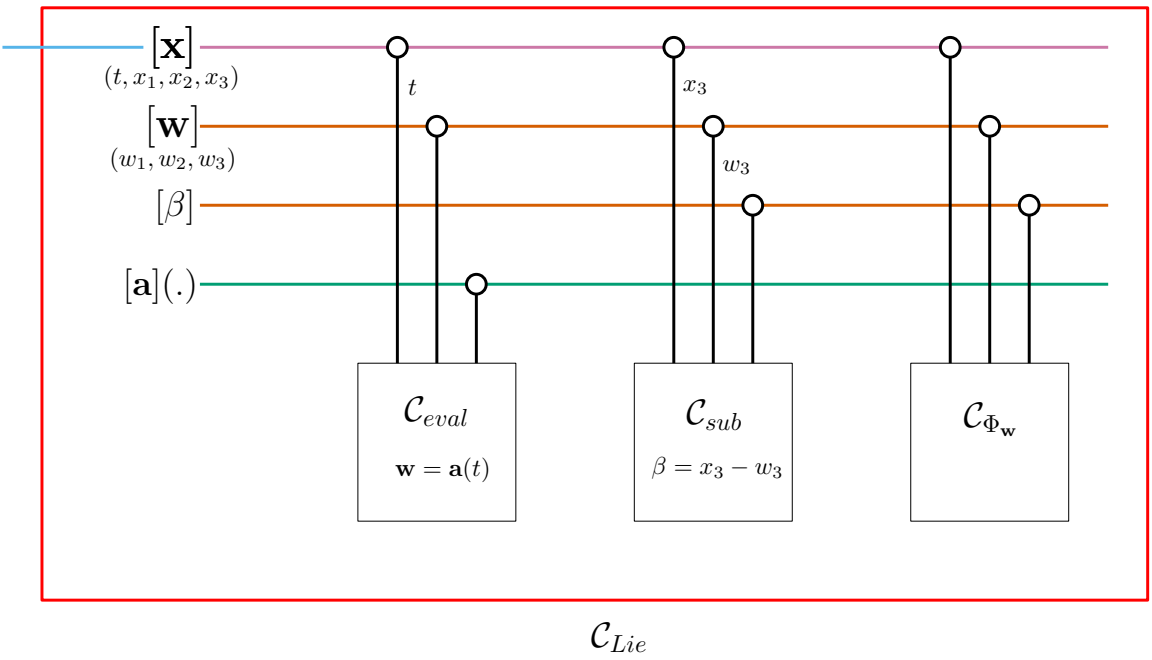


Figure B.2: Description of the $\mathcal{C}_{lie}$ presented in Figure B.1

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ALGORITHMS

BIBLIOGRAPHY

[1] Julien Alexandre dit Sandretto and Alexandre Chapoutot. "Validated Explicit and Implicit Runge-Kutta Methods". In: *Reliable Computing electronic edition*. Special Issue Devoted to Material Presented at SWIM 2015 22 (July 2016). URL: https://hal.archives-ouvertes.fr/hal-01243053 (visited on 12/22/2021).

[2] Matthias Althoff. "Reachability Analysis and Its Application to the Safety Assessment of Autonomous Cars". Technische Universität München, 2010.

[3] Ignacio Araya, Bertrand Neveu, and Gilles Trombettoni. "Exploiting Common Subexpressions in Numerical CSPs". In: *Principles and Practice of Constraint Programming*. Ed. by Peter J. Stuckey. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 342–357. ISBN: 978-3-540-85958-1. DOI: 10.1007/978-3-540-85958-1_23.

[4] E. Asarin, T. Dang, and O. Maler. "D/Dt: A Verification Tool for Hybrid Systems". In: *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*. Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228). Vol. 3. Dec. 2001, 2893–2898 vol.3. DOI: 10.1109/CDC.2001.980715.

[5] Jean-Pierre Aubin and Hèléne Frankowska. *Set-Valued Analysis*. Ed. by Jean-Pierre Aubin and Hélène Frankowska. Modern Birkhäuser Classics. Boston: Birkhäuser, 2009. ISBN: 978-0-8176-4848-0. DOI: 10.1007/978-0-8176-4848-0_1. URL: https://doi.org/10.1007/978-0-8176-4848-0_1 (visited on 01/27/2022).

[6] Stefan Banach. "Sur Les Opérations Dans Les Ensembles Abstraits et Leur Application Aux Équations Intégrales". In: *Fund. math* 3.1 (1922), pp. 133–181.

[7] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, July 31, 2017. 381 pp. ISBN: 978-1-108-50971-8. Google Books: 2SstDwAAQBAJ.

[8] Eckart Baumann. "Optimal Centered Forms". In: *BIT Numerical Mathematics* 28.1 (Mar. 1, 1988), pp. 80–87. ISSN: 1572-9125. DOI: 10.1007/BF01934696. URL: https://doi.org/10.1007/BF01934696 (visited on 01/05/2022).

[9] Aymeric Bethencourt and Luc Jaulin. "Cooperative Localization of Underwater Robots with Unsynchronized Clocks". In: *Paladyn: Journal of Behavioral Robotics* 4.4 (Dec. 2013), pp. 233–244. DOI: 10.2478/pjbr-2013-0023. URL: https://hal.archives-ouvertes.fr/hal-00989590 (visited on 12/21/2021).

[10] Aymeric Bethencourt and Luc Jaulin. "Solving Non-Linear Constraint Satisfaction Problems Involving Time-Dependant Functions". In: *Mathematics in Computer Science* 8.3 (Sept. 1, 2014), pp. 503–523. ISSN: 1661-8289. DOI: 10.1007/s11786-014-0209-6. URL: https://doi.org/10.1007/s11786-014-0209-6 (visited on 01/04/2022).

[11] Garrett Birkhoff. *Hydrodynamics—A Study in Logic, Fact and Similitude*. Princeton, NJ, USA: Princeton University Press, 1951.

[12] G. W. Bluman and J. D. Cole. *Similarity Methods for Differential Equations*. Vol. 13. Applied Mathematical Sciences. New York, NY: Springer-Verlag New York, 1974. 343 pp. ISBN: 978-1-4612-6394-4. Google Books: hyDSBwAAQBAJ.

[13] Silvère Bonnabel, Philippe Martin, and Pierre Rouchon. "Symmetry-Preserving Observers". In: *IEEE Transactions on Automatic Control* 53.11 (Dec. 2008), pp. 2514–2526. ISSN: 0018-9286. DOI: 10.1109/TAC.2008.2006929. URL: http://ieeexplore.ieee.org/document/4700863/ (visited on 07/20/2021).

[14] Auguste Bourgois. "Safe & Collaborative Autonomous Underwater Docking : Interval Methods for Proving the Feasibility of an Underwater Docking Problem". Theses. ENSTA Bretagne - École nationale supérieure de techniques avancées Bretagne, Jan. 2021. URL: https://tel.archives-ouvertes.fr/tel-03419041 (visited on 12/22/2021).

[15] Frédéric Boyer and Vincent Lebastard. "Exploration of Objects by an Underwater Robot with Electric Sense". In: *Biomimetic and Biohybrid Systems*. Ed. by Tony J. Prescott et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 50–61. ISBN: 978-3-642-31525-1. DOI: 10.1007/978-3-642-31525-1_5.

[16] Antoni Burguera, Yolanda González, and Gabriel Oliver. "The UspIC: Performing Scan Matching Localization Using an Imaging Sonar". In: *Sensors* 12.6 (June 2012), pp. 7855–7885. ISSN: 1424-8220. DOI: 10.3390/s120607855. URL: https://www.mdpi.com/1424-8220/12/6/7855 (visited on 03/18/2022).

[17] M. Ceberio and L. Granvilliers. "Horner's Rule for Interval Evaluation Revisited". In: *Computing. Archives for Scientific Computing* 69.1 (Sept. 2002), pp. 51–81. ISSN: 0010485X. DOI: 10.1007/s00607-002-1448-y. URL: http://link.springer.com/10.1007/s00607-002-1448-y (visited on 01/05/2022).

[18] Gilles Chabert. *The Ibex Library*. 2007. URL: http://www.ibex-lib.org/.

[19] Gilles Chabert and Luc Jaulin. "Contractor Programming". In: *Artificial Intelligence* 173.11 (July 2009), pp. 1079–1100. ISSN: 00043702. DOI: 10.1016/j.artint.2009.03.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0004370209000381 (visited on 11/29/2019).

[20] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 258–263. ISBN: 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_18.

[21] Alongkrit Chutinan and Bruce H. Krogh. "Verification of Polyhedral-Invariant Hybrid Automata Using Polygonal Flow Pipe Approximations". In: *Hybrid Systems: Computation and Control*. Ed. by Frits W. Vaandrager and Jan H. van Schuppen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 76–90. ISBN: 978-3-540-48983-2. DOI: 10.1007/3-540-48983-5_10.

[22] John G. Cleary. "LOGICAL ARITHMETIC". In: (Mar. 1, 1986). DOI: 10.11575/PRISM/30551. URL: https://prism.ucalgary.ca/handle/1880/45818 (visited on 12/16/2021).

[23] M. B. Clowes. "On Seeing Things". In: *Artificial Intelligence* 2.1 (Mar. 1, 1971), pp. 79–116. ISSN: 0004-3702. DOI: 10.1016/0004-3702(71)90005-1. URL: https://www.sciencedirect.com/science/article/pii/0004370271900051 (visited on 12/16/2021).

[24] Pieter Collins and Alexandre Goldsztejn. "The Reach-and-Evolve Algorithm for Reachability Analysis of Nonlinear Dynamical Systems". In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008) 223 (Dec. 26, 2008), pp. 87–102. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2008.12.033. URL: https://www.sciencedirect.com/science/article/pii/S1571066108004969 (visited on 11/03/2021).

[25] George Corliss and Robert Rihm. "Validating an A Priori Enclosure Using High-Order Taylor Series". In: *Mathematical Research* 90 (1996), pp. 228–238.

[26] Julien Damers, Luc JAULIN, and Simon Rohou. "Guaranteed Interval Integration for Large Initial Boxes". In: *Book of Abstracts of the 12th Summer Workshop on Interval Methods (SWIM 2019)*. SWIM 2019. Palaiseau, France: https://swim2019.ensta-paristech.fr/, July 2019.

[27] Julien Damers, Luc JAULIN, and Simon Rohou. "Lie Summetries Applied to Interval Integration (Provisonally Accepted)". In: *Automatica* (2022).

[28] Ernest Davis. "Constraint Propagation with Interval Labels". In: *Artificial Intelligence* 32.3 (July 1, 1987), pp. 281–331. ISSN: 0004-3702. DOI: 10.1016/0004-3702(87)90091-9. URL: https://www.sciencedirect.com/science/article/pii/0004370287900919 (visited on 12/16/2021).

[29] Nicolas Delanoue et al. "Guaranteed Characterization of Capture Basins of Nonlinear State-Space Systems". In: *Informatics in Control, Automation and Robotics: Selected Papers from the International Conference on Informatics in Control, Automation and Robotics 2007*. Ed. by Joaquim Filipe, Juan Andrade Cetto, and Jean-Louis Ferrier. Lecture Notes in Electrical Engineering. Berlin, Heidelberg: Springer, 2009, pp. 265–272. ISBN: 978-3-540-85640-5. DOI: 10.1007/978-3-540-85640-5_20. URL: https://doi.org/10.1007/978-3-540-85640-5_20 (visited on 04/25/2022).

[30] B. Desrochers and L. Jaulin. "A Minimal Contractor for the Polar Equation: Application to Robot Localization". In: *Engineering Applications of Artificial Intelligence* 55 (Oct. 1, 2016), pp. 83–92. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2016.06.005. URL: https://www.sciencedirect.com/science/article/pii/S0952197616301129 (visited on 12/16/2021).

[31] Derui Ding, Zidong Wang, and Qing-Long Han. "A Set-Membership Approach to Event-Triggered Filtering for General Nonlinear Systems Over Sensor Networks". In: *IEEE Transactions on Automatic Control* 65.4 (Apr. 2020), pp. 1792–1799. ISSN: 1558-2523. DOI: 10.1109/TAC.2019.2934389.

[32] L. E. Dubins. "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 (1957), pp. 497–516. ISSN: 0002-9327. DOI: 10.2307/2372560. JSTOR: 2372560.

[33] P.S Dwyer. "Computation with Approximate Numbers, Wiley, New York". In: *Linear Computations* (1951), pp. 11–34.

[34] P Eijgenraam. *The Solution of Initial Value Problems Using Interval Arithmetic : Formulation and Analysis of an Algorithm*. Centrum Voor Wiskunde en Informatica, Jan. 1, 1981. URL: https://ir.cwi.nl/pub/13004 (visited on 12/20/2021).

[35] T. F. Filippova et al. "Ellipsoidal State Estimation for Uncertain Dynamical Systems". In: *Bounding Approaches to System Identification*. Ed. by Mario Milanese et al. Boston, MA: Springer US, 1996, pp. 213–238. ISBN: 978-1-4757-9545-5. DOI: 10.1007/978-1-4757-9545-5_14. URL: https://doi.org/10.1007/978-1-4757-9545-5_14 (visited on 12/22/2021).

[36] Christian Forster et al. "On-Manifold Preintegration for Real-Time Visual–Inertial Odometry". In: *IEEE Transactions on Robotics* 33.1 (Feb. 2017), pp. 1–21. ISSN: 1941-0468. DOI: 10.1109/TRO.2016.2597321.

[37] Goran Frehse. "PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech". In: *International Journal on Software Tools for Technology Transfer* 10.3 (June 1, 2008), pp. 263–279. ISSN: 1433-2787. DOI: 10.1007/s10009-007-0062-x. URL: https://doi.org/10.1007/s10009-007-0062-x (visited on 02/28/2022).

[38] Eugene C. Freuder and Richard J. Wallace. "Partial Constraint Satisfaction". In: *Artificial Intelligence* 58.1 (Dec. 1, 1992), pp. 21–70. ISSN: 0004-3702. DOI: 10.1016/0004-3702(92)90004-H. URL: https://www.sciencedirect.com/science/article/pii/000437029290004H (visited on 12/17/2021).

[39] Z. Galias and P. Zgliczyński. "Computer Assisted Proof of Chaos in the Lorenz Equations". In: *Physica D: Nonlinear Phenomena* 115.3 (May 1, 1998), pp. 165–188. ISSN: 0167-2789. DOI: 10.1016/S0167-2789(97)00233-9. URL: https://www.sciencedirect.com/science/article/pii/S0167278997002339 (visited on 12/22/2021).

[40] Antoine Girard. "Computation and Stability Analysis of Limit Cycles in Piecewise Linear Hybrid Systems". In: *IFAC Proceedings Volumes*. IFAC Conference on Analysis and Design of Hybrid Systems 2003, St Malo, Brittany, France, 16-18 June 2003 36.6 (June 1, 2003), pp. 181–186. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)36428-5. URL: https://www.sciencedirect.com/science/article/pii/S1474667017364285 (visited on 01/11/2022).

[41] Antoine Girard. "Analyse Algorithmique des Systèmes Hybrides". PhD thesis. Institut National Polytechnique de Grenoble - INPG, Sept. 30, 2004. URL: https://tel.archives-ouvertes.fr/tel-00007064 (visited on 01/11/2022).

[42] Marco Giunti. *Computation, Dynamics, and Cognition*. Oxford University Press, June 26, 1997. 192 pp. ISBN: 978-0-19-535819-3.

[43] Tesjawi Gode. "Long Baseline Ranging Acoustic Positioning System". Virginia Tech, 2015.

[44] Kenneth Gold and Jacqueline C. Warsaw. "A Commemoration on the 50th Anniversary of the William Beebe-Otis Barton Bathysphere Dives". In: *Oceanography: The Past*. Ed. by Mary Sears and Daniel Merriman. New York, NY: Springer, 1980, pp. 393–396. ISBN: 978-1-4613-8090-0. DOI: 10.1007/978-1-4613-8090-0_37.

[45] Martin Golubitsky and Ian Stewart. *The Symmetry Perspective: From Equilibrium to Chaos in Phase Space and Physical Space*. Berlin: Birkhäuser, 2003. 348 pp. ISBN: 978-3-7643-2171-0. Google Books: 0HpyrroR9REC.

[46] Frédéric Goualard. *GAOL*. 2016. URL: https://github.com/goualard-f/GAOL (visited on 01/10/2022).

[47] Eric Goubault and Sylvie Putot. "Inner and Outer Reachability for the Verification of Control Systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19. New York, NY, USA: Association for Computing Machinery, Apr. 16, 2019, pp. 11–22. ISBN: 978-1-4503-6282-5. DOI: 10.1145/3302504.3311794. URL: https://doi.org/10.1145/3302504.3311794 (visited on 04/22/2022).

[48] Eric Goubault et al. "Inner Approximated Reachability Analysis". In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. HSCC '14. New York, NY, USA: Association for Computing Machinery, Apr. 15, 2014, pp. 163–172. ISBN: 978-1-4503-2732-9. DOI: 10.1145/2562059.2562113. URL: https://doi.org/10.1145/2562059.2562113 (visited on 01/27/2022).

[49] "Continuous-Time Dynamical Systems". In: *Optimal Control of Nonlinear Processes: With Applications in Drugs, Corruption, and Terror*. Ed. by Dieter Grass et al. Berlin, Heidelberg: Springer, 2008, pp. 9–98. ISBN: 978-3-540-77647-5. DOI: 10.1007/978-3-540-77647-5_2. URL: https://doi.org/10.1007/978-3-540-77647-5_2 (visited on 11/26/2021).

[50] The CAPD group. *CAPD - Computer Assisted Proofs in Dynamics, a Package for Rigorous Numerics*. 2005. URL: http://capd.ii.uj.edu.pl/ (visited on 01/04/2022).

[51] F. Hanson and S. Radic. "High Bandwidth Underwater Optical Communication". In: *Applied Optics* 47.2 (Jan. 10, 2008), pp. 277–283. ISSN: 1559-128X. DOI: 10.1364/AO.47.000277.

[52] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. "HyTech: A Model Checker for Hybrid Systems". In: *Computer Aided Verification*. Ed. by Orna Grumberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, pp. 460–463. ISBN: 978-3-540-69195-2. DOI: 10.1007/3-540-63166-6_48.

[53] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos (Third Edition)*. Ed. by Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. Boston: Academic Press, Jan. 1, 2013. ISBN: 978-0-12-382010-5. DOI: 10.1016/B978-0-12-382010-5.00007-5. URL: https://www.sciencedirect.com/science/article/pii/B9780123820105000075 (visited on 11/08/2021).

[54] Peter E. Hydon. *Symmetry Methods for Differential Equations: A Beginner's Guide*. Cambridge Texts in Applied Mathematics. Cambridge: Cambridge University Press, 2000. ISBN: 978-0-521-49703-9. DOI: 10.1017/CBO9780511623967. URL: https://www.cambridge.org/core/books/symmetry-methods-for-differential-equations/805DEEB7D5D0D2040A3A8444B1C8A33E (visited on 11/18/2021).

[55] Eero Hyvönen. "Constraint Reasoning Based on Interval Arithmetic". In: *IJCAI* (Jan. 1, 1989).

[56] IEA. "Evolution of Renewable Compared to Fossil Fuel Production in OECD Europe, 2010-2020". In: (Aug. 4, 2021). URL: https://www.iea.org/data-and-statistics/charts/evolution-of-renewable-compared-to-fossil-fuel-production-in-oecd-europe-2010-2020.

[57] Trachette Jackson and Ami Radunskaya. *Application of Dynamical Systems in Biology and Medicine*. Vol. 158. Springer, 2015.

[58] Luc Jaulin. "A Nonlinear Set Membership Approach for the Localization and Map Building of Underwater Robots". In: *IEEE Transactions on Robotics* 25.1 (Feb. 2009), pp. 88–98. ISSN: 1941-0468. DOI: 10.1109/TRO.2008.2010358.

[59] Luc Jaulin. "Robust Set-Membership State Estimation; Application to Underwater Robotics". In: *Automatica* 45.1 (Jan. 1, 2009), pp. 202–206. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2008.06.013. URL: https://www.sciencedirect.com/science/article/pii/S0005109808003853 (visited on 12/16/2021).

[60] Luc Jaulin. *Mobile Robotics*. John Wiley & Sons, Sept. 20, 2019. 391 pp. ISBN: 978-1-119-66349-2.

[61] Luc Jaulin and Benoît Desrochers. "Introduction to the Algebra of Separators with Application to Path Planning". In: *Engineering Applications of Artificial Intelligence* 33 (Aug. 1, 2014), pp. 141–147. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2014.04.010. URL: http://www.sciencedirect.com/science/article/pii/S0952197614000864 (visited on 12/11/2019).

[62] Luc Jaulin and Eric Walter. "Set Inversion via Interval Analysis for Nonlinear Bounded-Error Estimation". In: *Automatica* 29.4 (July 1, 1993), pp. 1053–1064. ISSN: 0005-1098. DOI: 10.1016/0005-1098(93)90106-4. URL: https://www.sciencedirect.com/science/article/pii/0005109893901064 (visited on 12/17/2021).

[63] Luc Jaulin et al. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Ed. by Luc Jaulin et al. London: Springer, 2001. ISBN: 978-1-4471-0249-6. DOI: 10.1007/978-1-4471-0249-6_2. URL: https://doi.org/10.1007/978-1-4471-0249-6_2 (visited on 12/10/2021).

[64] Luc Jaulin et al. "Interval Methods for Nonlinear Identification and Robust Control". In: 41st IEEE Conference on Decision and Control (CDC). Las Vegas, Nevada, USA, Dec. 2002.

[65] Tomasz Kapela, Piotr Zgliczyński, and ,Jagiellonian University, Institute of Computer Science, Łojasiewicza 6, 30-387 Kraków. "A Lohner-type Algorithm for Control Systems and Ordinary Differential Inclusions". In: *Discrete & Continuous Dynamical Systems - B* 11.2 (2009), pp. 365–385. ISSN: 1553-524X. DOI: 10.3934/dcdsb.2009.11.365. URL: http://aimsciences.org//article/doi/10.3934/dcdsb.2009.11.365 (visited on 06/01/2020).

[66] Tomasz Kapela et al. "CAPD::DynSys: A Flexible C++ Toolbox for Rigorous Numerical Analysis of Dynamical Systems". In: *Communications in Nonlinear Science and Numerical Simulation* 101 (Oct. 1, 2021), p. 105578. ISSN: 1007-5704. DOI: 10.1016/j.cnsns.2020.105578. URL: https://www.sciencedirect.com/science/article/pii/S1007570420304081 (visited on 01/04/2022).

[67] H. Kaushal and G. Kaddoum. "Underwater Optical Wireless Communication". In: *IEEE access : practical innovations, open solutions* 4 (2016), pp. 1518–1547. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2552538.

[68] Michel Kieffer et al. "Robust Autonomous Robot Localization Using Interval Analysis". In: *Reliable Computing* 6.3 (Aug. 1, 2000), pp. 337–362. ISSN: 1573-1340. DOI: 10.1023/A:1009990700281. URL: https://doi.org/10.1023/A:1009990700281 (visited on 04/27/2022).

[69] F Krückeberg. *Ordinary Differential Equations, Topics in Interval Analysis, Ed. E. Hansen*. Clarendron Press, Oxford, 1969.

[70] Vipin Kumar. "Algorithms for Constraint-Satisfaction Problems: A Survey". In: *AI Magazine* 13.1 (Mar. 15, 1992), pp. 32–32. ISSN: 2371-9621. DOI: 10.1609/aimag.v13i1.976. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/976 (visited on 12/16/2021).

[71] A. B. Kurzhanski and T. F. Filippova. "On the Theory of Trajectory Tubes — A Mathematical Formalism for Uncertain Dynamics, Viability and Control". In: *Advances in Nonlinear Dynamics and Control: A Report from Russia*. Ed. by Alexander B. Kurzhanski. Progress in Systems and Control Theory. Boston, MA: Birkhäuser, 1993, pp. 122–188. ISBN: 978-1-4612-0349-0. DOI: 10.1007/978-1-4612-0349-0_4. URL: https://doi.org/10.1007/978-1-4612-0349-0_4 (visited on 12/22/2021).

[72] Alexander B. Kurzhanski and Pravin Varaiya. "Ellipsoidal Techniques for Reachability Analysis". In: *Hybrid Systems: Computation and Control*. Ed. by Nancy Lynch and Bruce H. Krogh. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 202–214. ISBN: 978-3-540-46430-3. DOI: 10.1007/3-540-46430-1_19.

[73] Michel l'Hour, Vincent Creuze, and Denis Dégez. "Purpose-Built Robotic Tools and Methods for Deep-Sea Archaeology". In: IKUWA7 2020 - 7th International Congress on Underwater Archaeology. Vol. IKUWA7. June 2, 2020. URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-02388047 (visited on 04/22/2022).

[74] Hélène Lahanier, Eric Walter, and Roberto Gomeni. "OMNE: A New Robust Membership-Set Estimator for the Parameters of Nonlinear Models". In: *Journal of Pharmacokinetics and Biopharmaceutics* 15.2 (Apr. 1, 1987), pp. 203–219. ISSN: 0090-466X. DOI: 10.1007/BF01062344. URL: https://doi.org/10.1007/BF01062344 (visited on 04/29/2022).

[75] Lionel Lapierre et al. “Karst Exploration: Unconstrained Attitude Dynamic Control for an AUV”. In: *Ocean Engineering* 219 (Jan. 1, 2021), p. 108321. ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2020.108321. URL: https://www.sciencedirect.com/science/article/pii/S002980182031235X (visited on 04/22/2022).

[76] Cara E G LaPointe. “Virtual Long Baseline (VLBL) Autonomous Underwater Vehicle Navigation Using a Single Transponder”. Boston: Massachusetts Institute of Technology, 2006. 75 pp.

[77] Matheus Laranjeira, Claire Dune, and Vincent Hugel. “Embedded Visual Detection and Shape Identification of Underwater Umbilical for Vehicle Positioning”. In: *OCEANS 2019 - Marseille*. OCEANS 2019 - Marseille. June 2019, pp. 1–9. DOI: 10.1109/OCEANSE.2019.8867548.

[78] Fabrice Le Bars et al. “Set-Membership State Estimation with Fleeting Data”. In: *Automatica* 48.2 (Feb. 1, 2012), pp. 381–387. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2011.11.004. URL: http://www.sciencedirect.com/science/article/pii/S0005109811005322 (visited on 11/25/2019).

[79] Thomas Le Mézo. “Bracketing Largest Invariant Sets of Dynamical Systems: An Application to Drifting Underwater Robots in Ocean Currents”. Dec. 5, 2019.

[80] Sophus Lie. *Vorlesungen über differentialgleichungen: mit bekannten infinitesimalen transformationen*. B. G. Teubner, 1891. 592 pp. Google Books: FDcNAAAAYAAJ.

[81] Sophus Lie and F Engel. “Theorie Der Transformationsgruppen”. In: (1880).

[82] Rudolf J. Lohner. “Enclosing the Solutions of Ordinary Initial and Boundary Value Problems”. In: *Computerarithmetic* (1987), pp. 225–286.

[83] Alan K. Mackworth. “Consistency in Networks of Relations”. In: *Artificial Intelligence* 8.1 (Feb. 1, 1977), pp. 99–118. ISSN: 0004-3702. DOI: 10.1016/0004-3702(77)90007-8. URL: http://www.sciencedirect.com/science/article/pii/0004370277900078 (visited on 11/25/2019).

[84] Magne Mandt, Kenneth Gade, and Bjørn Jalving. “Integrating DGPS-USBL Position Measurements with Inertial Navigation in the HUGIN 3000 AUV”. In: *Proceedings from the 8th Saint Petersburg International Conference on Integrated Navigation Systems,* International Conference on Integrated Navigation Systems. Saint Petersburg, Russia, May 2001, p. 9.

[85] D. Meizel et al. “Initial Localization by Set Inversion”. In: *IEEE Transactions on Robotics and Automation* 18.6 (Dec. 2002), pp. 966–971. ISSN: 2374-958X. DOI: 10.1109/TRA.2002.805664.

[86] Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. *Interval Reachability Analysis: Bounding Trajectories of Uncertain Systems with Boxes for Control and Verification*. Springer Nature, Jan. 20, 2021. 115 pp. ISBN: 978-3-030-65110-7. Google Books: YG8WEAAAQBAJ.

[87] Mario Milanese and Carlo Novara. “Set Membership Identification of Nonlinear Systems”. In: *Automatica* 40.6 (June 1, 2004), pp. 957–975. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2004.02.002. URL: https://www.sciencedirect.com/science/article/pii/S0005109804000470 (visited on 04/29/2022).

[88] Ramon E Moore. "Automatic Local Coordinate Transformations to Reduce the Growth of Error Bounds in Interval Computation of Solutions of Ordinary Differential Equations". In: *Error in digital computation* 2 (1965), pp. 103–140.

[89] Ramon E Moore. "The Automatic Analysis and Control of Error in Digital Computation Based on the Use of Interval Numbers". In: *Error in digital computation* 1 (1965), pp. 61–130.

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

[91] Ramon E Moore, R Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. 1st ed. Society for Industrial and Applied Mathematics, 2009. ISBN: 978-0-89871-669-6.

[92] Nils Morozs, Paul Mitchell, and Yuriy V. Zakharov. "TDA-MAC: TDMA Without Clock Synchronization in Underwater Acoustic Networks". In: *IEEE access : practical innovations, open solutions* 6 (2018), pp. 1091–1108. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2777899.

[93] Nedialko S. Nedialkov and Kenneth R. Jackson. "An Interval Hermite-Obreschkoff Method for Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation". In: *Reliable Computing* 5.3 (Aug. 1, 1999), pp. 289–310. ISSN: 1573-1340. DOI: 10.1023/A:1009936607335. URL: https://doi.org/10.1023/A:1009936607335 (visited on 12/22/2021).

[94] Nedialko S. Nedialkov and Kenneth R. Jackson. *The Design and Implementation of an Object-Oriented Validated ODE Solver*. 2002.

[95] Nedialko S. Nedialkov, Kenneth R. Jackson, and John D. Pryce. "An Effective High-Order Interval Method for Validating Existence and Uniqueness of the Solution of an IVP for an ODE". In: *Reliable Computing* 7.6 (Dec. 1, 2001), pp. 449–465. ISSN: 1573-1340. DOI: 10.1023/A:1014798618404. URL: https://doi.org/10.1023/A:1014798618404 (visited on 01/04/2022).

[96] Peter J. Olver. *Applications of Lie Groups to Differential Equations*. 2nd ed. New York, NY: Springer, 2000. 548 pp. ISBN: 978-0-387-95000-6. Google Books: sI2bAxgLMXYC.

[97] L. V. Ovsiannikov. *Group Analysis of Differential Equations*. Academic Press, 1982. 433 pp. ISBN: 978-1-4832-1906-6. Google Books: YpPiBQAAQBAJ.

[98] Wuxu Peng. "Deadlock Detection in Communicating Finite State Machines by Even Reachability Analysis". In: *Mobile Networks and Applications* 2.3 (Nov. 1, 1997), pp. 251–257. ISSN: 1572-8153. DOI: 10.1023/A:1013640918785. URL: https://doi.org/10.1023/A:1013640918785 (visited on 02/28/2022).

[99] Nacim Ramdani and Nedialko S. Nedialkov. "Computing Reachable Sets for Uncertain Nonlinear Hybrid Systems Using Interval Constraint-Propagation Techniques". In: *Nonlinear Analysis: Hybrid Systems*. Special Issue Related to IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09) 5.2 (May 1, 2011), pp. 149–162. ISSN: 1751-570X. DOI: 10.1016/j.nahs.2010.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1751570X1000049X (visited on 02/23/2022).

[100] H. Ratschek. "Centered Forms". In: *SIAM Journal on Numerical Analysis* 17.5 (Oct. 1, 1980), pp. 656–662. ISSN: 0036-1429. DOI: 10.1137/0717055. URL: https://epubs.siam.org/doi/abs/10.1137/0717055 (visited on 01/05/2022).

[101] Andreas Rauh, Eberhard P. Hofer, and Ekaterina Auer. "VALENCIA-IVP: A Comparison with Other Initial Value Problem Solvers". In: *12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006)*. 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006). Sept. 2006, pp. 36–36. DOI: 10.1109/SCAN.2006.47.

[102] Simon Rohou. *Codac: Constraint-programming for Robotics*. 2020. URL: http://codac.io/.

[103] Simon Rohou et al. *Reliable Robot Localization: A Constraint-Programming Approach Over Dynamical Systems*. John Wiley & Sons, Dec. 5, 2019. 290 pp. ISBN: 978-1-119-68100-7. Google Books: Tx_CDwAAQBAJ.

[104] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, Aug. 18, 2006. 977 pp. ISBN: 978-0-08-046380-3. Google Books: Kjap9ZWcKOoC.

[105] Mohammad Taghi Sabet et al. "A Low-Cost Dead Reckoning Navigation System for an AUV Using a Robust AHRS: Design and Experimental Analysis". In: *IEEE Journal of Oceanic Engineering* 43.4 (Oct. 2018), pp. 927–939. ISSN: 1558-1691. DOI: 10.1109/JOE.2017.2769838.

[106] James T Sandefur. *Discrete Dynamical Systems: Theory and Applications*. Clarendon Press, 1990.

[107] Hermann Schichl and Arnold Neumaier. "Interval Analysis on Directed Acyclic Graphs for Global Optimization". In: *Journal of Global Optimization* 33.4 (Dec. 2005), pp. 541–562. ISSN: 0925-5001, 1573-2916. DOI: 10.1007/s10898-005-0937-x. URL: http://link.springer.com/10.1007/s10898-005-0937-x (visited on 01/09/2022).

[108] Jakob Schwichtenberg. *Physics from Symmetry*. Undergraduate Lecture Notes in Physics. Springer International Publishing, 2015. ISBN: 978-3-319-19201-7. DOI: 10.1007/978-3-319-19201-7. URL: https://www.springer.com/gp/book/9783319192017 (visited on 10/28/2020).

[109] Slobodan N. Simić et al. "Towards a Geometric Theory of Hybrid Systems". In: *Hybrid Systems: Computation and Control*. Ed. by Nancy Lynch and Bruce H. Krogh. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 421–436. ISBN: 978-3-540-46430-3. DOI: 10.1007/3-540-46430-1_35.

[110] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. "A Micro Lie Theory for State Estimation in Robotics". 2018. URL: http://arxiv.org/abs/1812.01537 (visited on 03/12/2021).

[111] Aage B. Sørensen. "Mathematical Models in Sociology". In: *Annual Review of Sociology* 4.1 (1978), pp. 345–371. DOI: 10.1146/annurev.so.04.080178.002021. URL: https://doi.org/10.1146/annurev.so.04.080178.002021 (visited on 01/11/2022).

[112] John Stillwell. *Naive Lie Theory*. Ed. by John Stillwell. Undergraduate Texts in Mathematics. New York, NY: Springer, 2008. ISBN: 978-0-387-78215-7. DOI: 10.1007/978-0-387-78214-0_3. URL: https://doi.org/10.1007/978-0-387-78214-0_3 (visited on 01/09/2022).

[113] Teruo Sunaga. "Theory of Interval Algebra and Its Application to Numerical Analysis". In: *RAAG memoirs* 2 (1958), pp. 29–58.

[114] TeleGeography. *Submarine Cable Map*. Sept. 3, 2022. URL: https://www.submarinecablemap.com/ (visited on 03/15/2022).

[115] L. D Thomas et al. "Comparison of Numerical Methods for Solving the Second-Order Differential Equations of Molecular Scattering Theory". In: *Journal of Computational Physics* 41.2 (June 1, 1981), pp. 407–426. ISSN: 0021-9991. DOI: 10.1016/0021-9991(81)90103-0. URL: https://www.sciencedirect.com/science/article/pii/0021999181901030 (visited on 01/11/2022).

[116] S. Tornil, T. Escobet, and L. Travé-Massuyès. "Robust Fault Detection Using Interval Models". In: *2003 European Control Conference (ECC)*. 2003 European Control Conference (ECC). Sept. 2003, pp. 1003–1008. DOI: 10.23919/ECC.2003.7085090.

[117] Warwick Tucker. "The Lorenz Attractor Exists". In: *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 328.12 (June 15, 1999), pp. 1197–1202. ISSN: 0764-4442. DOI: 10.1016/S0764-4442(99)80439-X. URL: https://www.sciencedirect.com/science/article/pii/S076444429980439X (visited on 12/20/2021).

[118] David L. Waltz. "Generating Semantic Descriptions From Drawings of Scenes With Shadows". In: (Nov. 1, 1972). URL: https://dspace.mit.edu/handle/1721.1/6911 (visited on 12/16/2021).

[119] R A Wijsman. "Cross-Sections of Orbits and Their Application to Densities of Maximal Invariants". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. University of California press, 1967, pp. 389–400.

[120] Daniel Wilczak and Piotr Zgliczyński. "Cr-Lohner algorithm". In: *Schedae Informaticae* 2011 (Volume 20 Jan. 23, 2012), pp. 9–42. ISSN: 2083-8476. DOI: 10.4467/20838476SI.11.001.0287. URL: https://www.ejournals.eu/Schedae-Informaticae/2011/Volume-20/art/1201/ (visited on 12/22/2021).

[121] Isaak Moiseevich Yaglom. *Felix Klein and Sophus Lie: Evolution of the Idea of Symmetry in the Nineteenth Century*. Boston, MA: Birkhäuser, 1988.

[122] Zheping Yan et al. "Research on an Improved Dead Reckoning for AUV Navigation". In: *2010 Chinese Control and Decision Conference*. 2010 Chinese Control and Decision Conference. May 2010, pp. 1793–1797. DOI: 10.1109/CCDC.2010.5498571.

[123] Xi Yu et al. "Scheduling Multiple Agents in a Persistent Monitoring Task Using Reachability Analysis". In: *IEEE Transactions on Automatic Control* 65.4 (Apr. 2020), pp. 1499–1513. ISSN: 1558-2523. DOI: 10.1109/TAC.2019.2922506.

[124] Piotr Zgliczynski. "C1 Lohner Algorithm". In: *Foundations of Computational Mathematics* 4.2 (2002), pp. 429–465. ISSN: 1615-3375, 1615-3383. DOI: 10.1007/s102080010025. URL: https://www.infona.pl//resource/bwmeta1.element.springer-ad0a9700-671c-30c8-a9bf-3b331ebb7d6e (visited on 12/22/2021).

[125] Wei-bin Zhang. *Differential Equations, Bifurcations And Chaos In Economics*. World Scientific Publishing Company, July 18, 2005. 510 pp. ISBN: 978-981-310-651-2. Google Books: zNY7DQAAQBAJ.