DOCTORAT BRETAGNE LOIRE MATHSTIC

ENSTA Bretagne

FORSSEA ROBOTICS

Lab-STICC

THÈSE DE DOCTORAT DE

ENSTA BRETAGNE

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

Par

Auguste BOURGOIS

Safe & collaborative autonomous underwater docking

Interval methods for proving the feasibility of an underwater docking problem

Thèse présentée et soutenue à Brest, le 20 Janvier 2021
Unité de recherche : Lab-STICC
Thèse N° : 2017/0556

Rapporteurs avant soutenance :

Sylvie PUTOT	Professeur, École Polytechnique, Palaiseau, FR
Daniel WILCZAK	Professeur, Jagiellonian University, Kraków, PL

Composition du Jury :

Président :	Gilles TROMBETTONI	Professeur, LIRMM, Montpellier, FR
Examinateurs :	Claire DUNE	Maître de conférence, COSMER, Toulon, FR
	Andreas RAUH	Enseignant-chercheur, ENSTA Bretagne, Brest ,FR
	Simon ROHOU	Enseignant-chercheur, ENSTA Bretagne, Brest, FR
	Benoît ZERR	Professeur, ENSTA Bretagne, Brest, FR
Dir. de thèse :	Luc JAULIN	Professeur, Lab-STICC, Brest, FR

Invité(s) :

Gautier DREYFUS	CEO, Forssea Robotics, Paris, FR

SAFE & COLLABORATIVE AUTONOMOUS UNDERWATER DOCKING

Interval methods for proving feasibility of an underwater docking problem

AUGUSTE BOURGOIS

Brest, January 20th, 2021

Auguste Bourgois: *Safe & collaborative autonomous underwater docking,* Interval methods for proving feasibility of an underwater docking problem, © January 20th, 2021

À ma famille.

Never lose sight of the role your particular subject has within the great performance of the tragicomedy of human life; keep in touch with life – not so much with practical life as with the ideal background of life, which is ever so much more important; and, Keep life in touch with you. *If you cannot – in the long run – tell everyone what you have been doing, your doing has been worthless.*

— Erwin Schrödinger [110]

ACKNOWLEDGMENTS

Ce manuscrit marque la fin de trois ans de travail, caractérisés par une dualité entre recherche académique, théorique et abstraite au possible, et ingénierie de conception et de terrain. Réussir à mener ces deux activités de front n'aurait pas été possible sans l'aide et le soutien de nombreuses personnes que je souhaite remercier ici.

Tout d'abord, merci au professeur Luc Jaulin, mon directeur de thèse, pour son flot continu d'idées nouvelles, sans lesquelles cette thèse n'aurait pas vu le jour, son attention dans les moments difficiles et sa patience face à mon expression circonspecte quand il me présentait un nouveau raisonnement mathématique. Je tiens aussi à le remercier pour son extraordinaire disponibilité, qualité rare chez les directeurs de thèse.

Je suis aussi particulièrement reconnaissant à Gautier Dreyfus, président de Forssea Robotics, de m'avoir fait confiance dès le premier jour et tout au long de ces trois ans, de m'avoir permis de participer à des projets ambitieux et intéressants, allant des premières étapes de conception d'un robot à ses essais en mer et de s'être montré particulièrement compréhensif quant au temps nécessaire pour mener à bien des travaux académiques.

Ensuite, je remercie les membres de mon jury de thèse, réunis tant bien que mal du fait de la crise sanitaire, d'avoir été disponibles et de s'être montrés intéressés par mon travail : *I would like to thank Sylvie Putot and Daniel Wilczak, who did me the honour of reviewing my thesis, and significantly contributed to its improvement. A special thank you to Daniel Wilczak for his support with the CAPD library and its inner algorithms.* Merci à Gilles Trombettoni d'avoir accepté de présider mon jury et à Claire Dune pour son accompagnement. Merci à Andreas Rauh pour nos échanges particulièrement intéressants à propos de l'analyse de stabilité et des pistes d'amélioration et d'application de la méthode développée dans ce document. Merci à Benoît Zerr de m'avoir aidé et supporté non seulement tout au long de cette thèse, mais plus

généralement depuis ma première année à l'ENSTA Bretagne. Enfin, merci à Simon Rohou pour sa relecture attentive et intéressée de mon manuscrit et pour ses conseils particulièrement utiles jusqu'à cinq minutes avant la soutenance. Benoît, Simon, merci à vous deux pour l'enthousiasme communicatif inspirant dont vous faites preuve dans votre travail quotidien.

Naturellement, je suis très reconnaissant envers Annick Billon-Coat pour ses qualités organisationnelles, en particulier de dernière minute pour la préparation de la soutenance, ainsi que pour le suivi administratif dont elle m'a fait bénéficier pendant ces trois ans, ce qui est une tâche particulièrement ardue.

Ensuite, j'aimerais remercier chaleureusement mes camarades doctorants, post-doctorants et ingénieurs de recherche à l'ENSTA Bretagne, d'une part pour m'avoir aidé, motivé et supporté avant, pendant (et après) la rédaction, mais aussi pour toutes les activités "extra" que nous avons faites ensemble (voile, wakeboard, escalade, randonnées, crêpes, raclettes, fondues, plage, concerts...) : merci à Thomas Le Mezo, Julien Damers, Joris Tillet, Irène Mopin, Yoann Sola, Romain Schwab, Marie Ponchart, Fabien Novella, Maël Le Gallic. J'en profite pour souhaiter bon courage aux futurs docteurs. Je remercie spécialement Julien, pour son travail de relecture, et Thomas, pour sa participation à la préparation de ma soutenance. Je remercie aussi les autres membres du laboratoire, en particulier Fabrice Le Bars, Alain Bertholom, Didier Tanguy, Gilles Le Maillot, Michel Legris, que j'ai beaucoup écoutés, et grâce auxquels j'ai beaucoup appris.

Bien sûr, je tiens aussi à remercier toute l'équipe de Forssea Robotics, pour avoir été des collègues formidables, pour les échanges passionnants (et passionnés) que nous avons eu, pour les déjeuners et les dîners en équipe, et pour toutes les choses que j'ai apprises à vos côtés. Ce fût un véritable plaisir de travailler avec vous tous (et ce n'est pas fini) : Alaa El Jawad, David Barral, Alexis Azoura, Jaouad Hajjami, Manuel Lopes Mendes, Mathieu Mège, Ian McElroy, Laurent Tardivon, Florent Taurines. Un merci tout spécial à Alaa, pour notre fructueuse collaboration, due à la similarité de notre conception de ce qu'est une *belle* architecture, ainsi qu'à notre amour partagé pour la destruction (et la reconstruction).

Je remercie aussi mes amis de m'avoir supporté, d'avoir suivi avec intérêt le déroulement de cette thèse, et d'avoir assisté à la soutenance. Merci pour tous les bons moments passés et à venir.

A very special thank you to Princess Anna Lingner, who has been standing by my side for more than two years, proofreading my work and correcting my English. In particular, she and her mother helped me obtaining a version of the original paper from Lohner [75] from the archives of a German university, which significantly contributed to my understanding of his algorithm.

Par dessus tout, je tiens à remercier ma famille, en commençant par mes parents, pour leur affection sans bornes, leur soutien indéfectible, leurs encouragements énergiques, et leur aide pour l'organisation de la soutenance ; ainsi que mes frères et sœur pour m'avoir supporté (plutôt dans le deuxième sens du terme, mais c'est réciproque) pendant 25 ans, et surtout ces trois dernières années. Merci à mes grands-parents, mes oncles et tantes et mes cousins pour l'intérêt porté à mes études et leurs messages d'encouragement, et merci à ceux qui ont pu assister à la soutenance. Pierre-Alain, merci de m'avoir montré la voie.

Enfin, je remercie le futur lecteur de ces pages pour l'intérêt porté au travail qui m'a occupé pendant trois ans. J'espère qu'elles lui permettront d'obtenir un premier aperçu des concepts de systèmes dynamiques, de stabilité et d'intégration garantie. J'aurai essayé, dans une certaine mesure, d'appliquer le conseil de Schrödinger.

CONTENTS

LIST OF FIGURES

LIST OF PROGRAMS

LIST OF ALGORITHMS

ACRONYMS

ROV	Remotely Operated Vehicle
AUV	Autonomous Underwater Vehicle
USV	Unmanned Surface Vehicle
CAPD	Computer Assisted Proofs in Dynamics
IVP	Initial Value Problem
CSP	Constraint Satisfaction Problem
IMU	Inertial Measurement Unit
GPS	Global Positioning System
EKF	Extended Kalman Filter

NOTATIONS

Dynamical systems

$\phi(\dots,\dots)$	Flow function of a dynamical system

General notations

x, α, T	Scalar values
$\mathbf{x} = (x,y,z) = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	Vector value (column)
$\mathbf{X} = \begin{bmatrix} x_{ii} & x_{ij} \\ x_{ji} & x_{jj} \end{bmatrix}$	Matrix value
$f(\dots)$	Scalar-valued function
$\mathbf{f}(\dots)$	Vector-valued function
$\mathbf{F}(\dots)$	Matrix-valued function
$\mathbf{J_f}(\dots)$	Jacobian matrix of a function $\mathbf{f}$
$\mathbf{0}$	Null vector, sometimes referred to as *the origin*
$\mathbf{0}_n$	n-dimentional null vector, sometimes referred to as *the origin*
$\mathbf{I}_n$	n-dimentional identity matrix

Intervals

$[x]$	Interval of $\mathbb{R}$
$[x]^{\times n}$	Interval of $\mathbb{R}^n$, with all components initialised at $[x]$
$[\mathbf{x}]$	Box of $\mathbb{R}^n$
$[\mathbf{X}]$	Interval matrix of $\mathbb{R}^{n\times m}$
$\mathbf{f}([\mathbf{x}])$	Image set of $[\mathbf{x}]$ by $\mathbf{f}$
$[\mathbf{f}([\mathbf{x}])]$	Minimal inclusion function for $\mathbf{f}([\mathbf{x}])$
$[\mathbf{f}]([\mathbf{x}])$	Inclusion function for $\mathbf{f}([\mathbf{x}])$

$[\mathbf{f}_c]([\mathbf{x}])$	Centred form inclusion function for $\mathbf{f}([\mathbf{x}])$
$\mathcal{L}_f$	Constraint defined by the equation $f(\mathbf{x}) = 0$
$\mathcal{C}_f$	Contractor associated with the constraint $\mathcal{L}_f$

Sets

$\mathcal{X}$	Mathematical set
$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integers
$\mathbb{IR}$	Set of intervals of $\mathbb{R}$
$\mathbb{IR}^n$	Set of axis-aligned boxes of $\mathbb{R}^n$
$\mathcal{C}^m(\mathbb{R}^n)$	Set functions of $\mathbb{R}^n$ with m continuous derivatives
$\emptyset$	Empty set

Tubes

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

1

INTRODUCTION

TABLE OF CONTENTS

1.1 CONTEXT

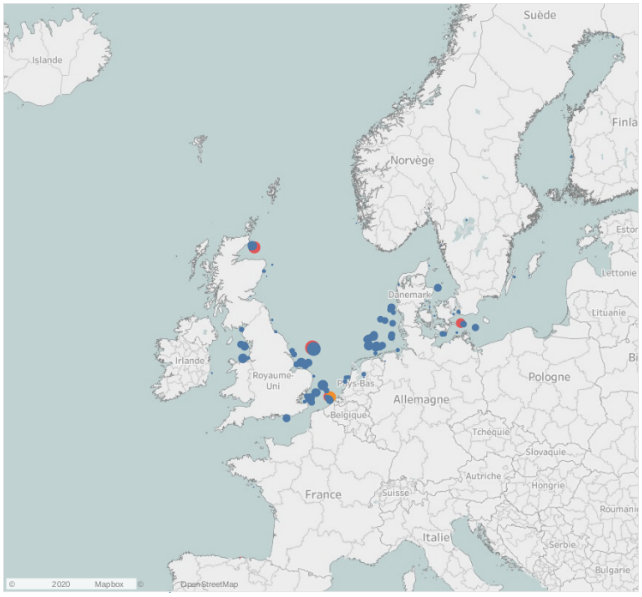
1.1.1 *And there was light*

Our civilisation runs on energy. This fact has never been more true than today: between 1973 and 2018, the world's total energy consumption has more than doubled, according to the recent annual report issued by the *International Energy Agency* [44]. However, this rise in energy consumption comes with that of CO_2 emissions, among other environmental issues. Most countries of the world are thus increasingly looking towards renewable energy as a serious alternative to traditional fossil fuels and nuclear energy: in 2019, 75% of new power generating installations were using renewable energies, while this share was of less than 50% in 2009, according to the report published recently by the *Renewable Energy Policy Network for the 21st Century* [86].

Renewable energy comes from different sources. The most developed today are hydropower, wind (onshore and offshore) and solar photovoltaic power: in 10 years, the wind power production capacity has more than quadrupled, and this trend is accelerating. In particular, offshore wind power production, despite representing only 5% of the total wind power capacity, accounted for 10% of the new installations

An extensive list of offshore wind farms is available on the 4C Offshore website (https://www.4coffshore.com/windfarms/)

in 2019. One can think of the newly installed wind farms in the North Sea or off the Chinese coast. As of today, more than 3000 offshore wind turbines have been deployed, and at least a thousand more will be installed in the next two years. Another lesser-known source of renewable energy is ocean power (understand the body of water), despite the latter's enormous potential: energy production via exploitation of tidal streams and waves are being studied from Europe to Northern America and China. While these projects are still in the early stages of development, investments in that domain are steadily increasing, reflecting the growing interest of governments and industries in ocean power.



Country	Details	
	MW connected	Turbines connected
BELGIUM	1 791	343
DENMARK	1 703	559
FINLAND	71	19
FRANCE	2	1
GERMANY	7 659	1 501
IRELAND	25	7
NETHERLANDS	1 342	393
NORWAY	2	1
PORTUGAL	25	3
SPAIN	5	1
SWEDEN	192	80
UK	10 428	2 294

Figure 1.1: Map displaying the existing offshore wind farms in the North Sea (https://windeurope.org/about-wind/interactive-offshore-maps/)

These installations, however, require more inspection and maintenance operations than their inshore counterparts: pressure, corrosion, marine life (in particular the so-called fouling, a mix of algae, barnacles and other marine organisms), and industrial waste are a threat to their integrity. Until recently, inspection and maintenance operations of offshore installations, such as oil platforms, were realised thanks to a combination of highly skilled divers in diving suits and robots piloted from the surface (usually known as Remotely Operated Vehicle (ROV)). However, because of the risks of deep diving in an industrial environment, and because of the size of the boat required to launch, operate and recover these robots, the inspection and maintenance costs are particularly high. Therefore, the growing number of offshore installations triggered the need for cheaper inspection and maintenance processes.

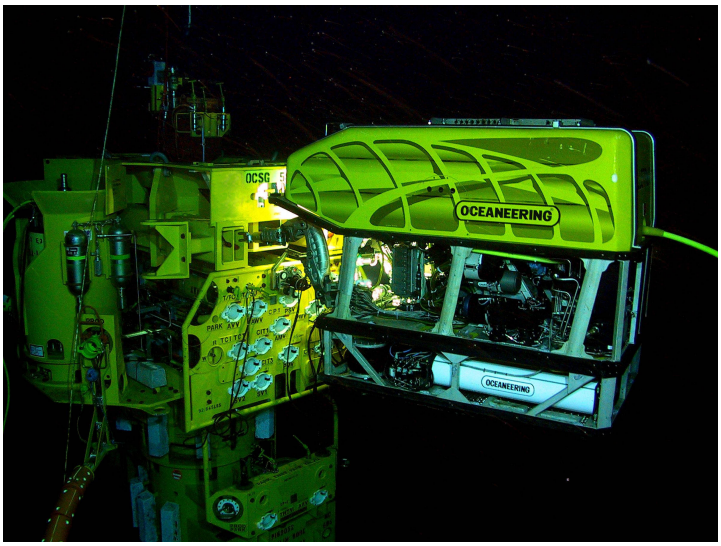


Figure 1.2: Maintenance operation on an underwater structure by a work class ROV (Released by Oceaneering in the public domain)

1.1.2 *Dawn of the robots*

As mentioned earlier, the robots used for maintenance operations are called ROVs. In short, a robot is launched by a surface vessel, to which it remains linked via a tether for power and communication [7], [19]. The main reasons for using a ROV instead of an Autonomous Underwater Vehicle (AUV) are the power demand during these operations that is usually too high for batteries to sustain, and sometimes the complexity of the mission (soldering pipes, guiding the installation of a new wellhead...). They are usually classified into two categories [19, Chapter 1], [25]:

- *Work class* ROVs weigh between a few hundred kilograms to a few tons. They are usually equipped with various tools such as robotic arms, probes, rotating brushes, spotlights... The latter are often powered using a hydraulic pump, and so are the thrusters. They embed a variety of classical sensors: Inertial Measurement Unit (IMU), pressure sensor, cameras, altimeter, Doppler Velocity Log (DVL), sonars, and more sophisticated acoustic localisation systems (usually a transducer to interact with Long Base Line (LBL) transponders or a transponder to interact with a vessel mounted Ultra Short Base Line (USBL) antenna). Their missions usually consist in heavy maintenance or deployment operations on the ocean floor.
- *Observation class* ROVs are lighter than the work class ones (usually less than 300 kg). They embed fewer pieces of equipment, usually a couple of cameras and spotlights, a Conductivity-Temperature-Depth (CTD) probe and a reduced set of sensors

> for localisation purposes. Their missions typically fall in the inspection category: checking an underwater pipeline or the wall of a dam, exploring a wreck... They can sometimes be used for light maintenance operations, such as the cleaning of underwater structures.

Operating such heavy equipment requires major financial, material and human resources. First of all, deploying a ROV is only achievable from a boat, equipped with a Launch And Recovery System (LARS) and a power generator (except for the micro ROVs embedding batteries, which can be launched manually). Secondly, an acoustic positioning system is used most of the time, either a LBL, which requires the deployment and the calibration of acoustic transponders underwater, or an USBL, which requires an antenna adequately calibrated on the surface vessel and a transponder on the robot. Thirdly, a Tether Management System (TMS), and a clump weight must be used, depending on the operating depth and the robot's mass and power. Indeed, a small robot cannot counter the tether's drag when diving too deep. In that case, the clump weight tenses the tether, and the TMS controls the length being released or rewound. The most sophisticated versions of these pieces of equipment are automated to some extent: the ROV usually provides semi-automated manual control (e. g. for station keeping or maintaining depth), while the TMS can mitigate the swell-induced jerks in the tether.

Figure 1.3 shows the different steps of a ROV operation, and some of the equipment mentioned above can be spotted on the pictures.

1.1.3 *Towards autonomous ROVs*

In this context of offshore installation growth, keeping such operation procedures is therefore very costly in terms of time, human, material and financial resources. Thus, offshore companies have started investing time and money in research and development programs towards task automation, particularly inspection missions. Multiple approaches are being considered:

- Using *resident ROVs*, whether autonomous or not, to perform inspection missions. Resident ROVs are deployed once and parked in a garage at the bottom of the sea. These garages embed a TMS, and are linked to the shore to communicate with and power the robot. The latter can then be operated directly from the shore without the need for a surface vessel, and regardless of the weather conditions.
- Using a ROV deployed from a Unmanned Surface Vehicle (USV). The latter's goal is to enable powering and communicating with the robot from the surface (and from the shore via satellites or a

(a) The LARS launches Argos and its garage

(b) The tether is managed manually by a crew member

(c) Argos overhanging a plane wreck by 40 m depth, linked to the surface by its tether

Figure 1.3: Pictures taken during sea trials of the observation class ROV Argos (Courtesy of Forssea Robotics/Balao)

more common data network) while reducing the costs involved by a larger vessel, mainly due to the crew wages and the fuel consumption.

- Using a smaller than usual ROV deployed from a lighter than usual surface vessel; the robot, the launch and recovery manoeuvres ideally being automated. Doing so also reduces operational expenses while providing the security of having humans ready to intervene, should any problem arise.

Variants of the two last scenarios also involve a garage hanging from the surface unit to allow for easier recovery.

1.1.4 *Current issues in autonomous underwater robotics*

Creating an autonomous underwater robot is no easy task. Multiple constraints have to be taken into account [19, Chapter 2]. Since the

robot will be evolving underwater, the physical constraints inherent to this environment must be considered: pressure, corrosion, darkness, electrochemical reactions, temperature; all these force the constructors to choose the materials used on the robot carefully. The latter's design must also account for the possibly rough operating conditions, involving shocks, vibrations, and potentially careless handling by operators.

1.1.4.1 *Positioning of autonomous underwater robots*

Unlike in terrestrial and aerial robotics, positioning a robot underwater is quite tricky. While pressure sensors allow computing the robot's depth quite precisely, there are no equivalent sensors for the robot's horizontal position. There is no such thing as an underwater Global Positioning System (GPS): positioning systems exist for underwater applications (we already mentioned LBL and USBL acoustic systems), but they are much more expensive, usually less precise, and require specific calibration processes. In their absence, underwater robots generally rely on an IMU (which can be – and ought to be – much more precise than their aerial and terrestrial counterparts), the acceleration and angular speed of which are integrated over time to obtain a rough approximation of the robot's position. The latter can be enhanced by using other sensors, such as the DVL, which measures the robot's relative speed with respect to the ground, or the surrounding body of water.

Cameras can be used to localise the robot with respect to a nearby obstacle [29], [128], [130]. However, cameras are of little help in detecting points of interest or obstacles farther than a few meters because of the surrounding darkness passed a hundred meters depth and water turbidity. Additionally, the lack of identifiable natural visual markers is problematic in some applications. Special techniques such as polarised light and lenses, adapted filtering algorithms, and artificial markers ought to be used to achieve practical results [39], [106], [129]. Sonars can be used, especially since the appearance of small, low-cost units. However, in both cases, the data's online processing is computationally expensive, involving delays in the positioning algorithms and a relatively low measurement frequency.

Argos, the autonomous observation class ROV developed by Forssea Robotics (see Figure 1.4), embeds the sensors mentioned above.

1.1.4.2 *Accuracy of localisation & control algorithms*

The accuracy and the precision of a robot's estimated position depend on its sensors and positioning strategy. Measurements can be of two different types: *proprioceptive*, in which case the sensor is self-contained

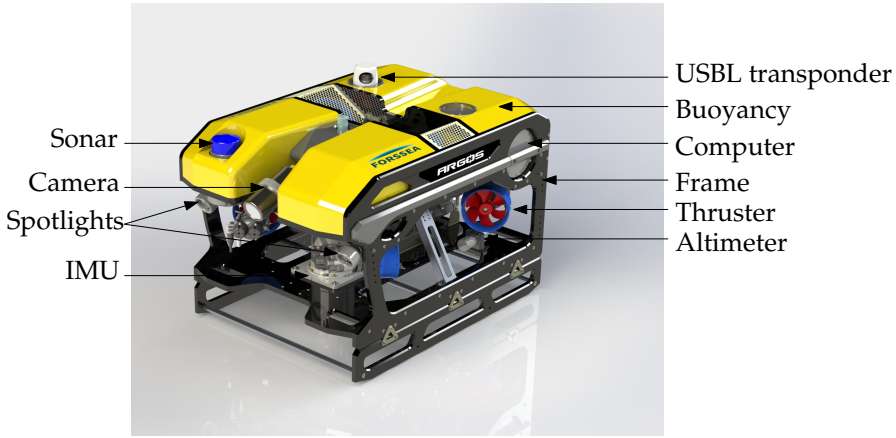


Figure 1.4: Argos, an autonomous ROV developed by Forssea Robotics (Courtesy of Forssea Robotics)

and measures its own dynamical characteristics (acceleration, rotational speed for example); and *exteroceptive*, when the sensor measures a quantity linked to the environment (pressure, light, acoustic waves...).

Usually, the robot's computer embeds a predictor-corrector algorithm for localisation. In the absence of exteroceptive measurements, the robot evolves in *dead-reckoning*. Its position is estimated by integrating proprioceptive data. Therefore, it can drift, and its accuracy decreases quickly with time, depending on the offsets and noise of the sensors and the associated filtering algorithm. This corresponds to the predictor part of the localisation algorithm. When exteroceptive measurements are performed, the corrector comes into play: the prediction is computed using the newly obtained data, and if the precision and accuracy of the data are high enough, the ones of the estimated position will be adjusted. These predictor-corrector algorithms are most of the time implemented using a flavour of *Kalman filter* [48, Chapter 7], [121, Chapter 3], [111, Chapter 4]. These algorithms usually require mathematical models of the robot and its sensors. However, the parameters of the latter are not necessarily perfectly known.

Other factors can influence the precision of a robot's estimated position, in particular underwater, in a dead reckoning scenario:

- underwater currents can reach a few meters per second;
- in shallow water, the swell induces circular translation of the body of water and of what it encompasses;
- the drag caused by the tether can be non-negligible, in particular for small ROVs. The authors of [65], [66] proposed a solution to that problem in the form of an actionable tether.

Therefore, the estimated position of a robot is uncertain. The control algorithms then process uncertain data, outputting uncertain commands, which results in an uncertain robot's behaviour.

1.2 AUTONOMOUS UNDERWATER DOCKING

In Section 1.1.3, we presented the three different scenarios envisioned by offshore companies to increase automation in their inspection and maintenance processes. What characterises the latter is that in each case, the robot needs to cruise back autonomously to a specific area to be recovered, either a garage, the moon pool of a USV, or the recovering area of a surface vessel. In other words, the robot must perform a *docking* manoeuvre onto a specific target, whether it is the centre point of a specific imaginary area, an underwater garage, or an energy outlet to recharge its batteries. A well known, recent example of an autonomous docking manoeuvre is the one performed by new-generation cargo ships on the International Space Station (ISS) (see Figure 1.5).

Figure 1.5: A SpaceX Dragon performing a docking manoeuvre with the ISS (Copyright free picture from NASA)

1.2.1 *Literature review*

The subject of autonomous underwater docking started mushrooming in the early nineties when the required technologies (control & localisation algorithms, computational power...) reached a sufficiently mature stage. Different approaches have been proposed: while the most common one is based on a physical or imaginary funnel in which the robot must enter [115], [117], [126], other designs include string-

based systems (the robot has to catch a string, which then pulls the former towards the docking station) [33], [64], [114] or grasping-based docking (an embedded robotic arm drives the docking manoeuvre) [30], [91]. Usually, a docking mission is split into two phases [58], [61], [117], [126]: the long-range, rough approach, during which the robot must join a specific area located near its docking target; and a short-range, precise approach which is usually triggered only if all the safety conditions and localisation requirements are met. While the robot mainly uses acoustic localisation systems or dead reckoning during the long-range approach, it switches to more precise sensors for the final manoeuvre: cameras aiming for lights or visual markers is usually the preferred solution [24], [61], [77], [91], [122], [126] although other methods such as electromagnetic [31] or electric field [13] guidance have been studied . Chosen control methods range from classical PID controllers [58], [91] to fuzzy logic [93], [115].

Therefore, many different approaches do exist to complete an autonomous docking mission underwater. At Forssea Robotics, the choice has been made to orientate the efforts towards the "two-phases" approach, using visual markers detection as a short-range localisation strategy. A classical chain of command controls the robot (see Figure 1.6), and the garage has a funnel shape to ease the manoeuvre (see Figure 1.7).

1.2.2 *Research approach*

As we highlighted earlier, there are many different methods to perform a docking mission: differences lie in the docking station, the chosen controller or localisation method. All of these methods have proved to be efficient to some extent. However, to deploy an entirely autonomous ROV and recover it through a docking station demands more guarantees. Indeed, it might be possible that the ROV cannot dock into its garage because of some external events, or that the chosen control method fails in certain conditions, or that the localisation algorithm does not indicate the correct position of the robot. Therefore, there is a need for a validation method that could prove feasibility of a mission, and in our case of a docking manoeuvre. In this thesis, we thus chose to work on validation methods applied to the docking problem.

1.2.2.1 *Presentation of the approach*

An autonomous ROV is a complex machine, embedding various sensors measuring data from the environment, and actuators acting on the latter. All these pieces of equipment are controlled by a computer,

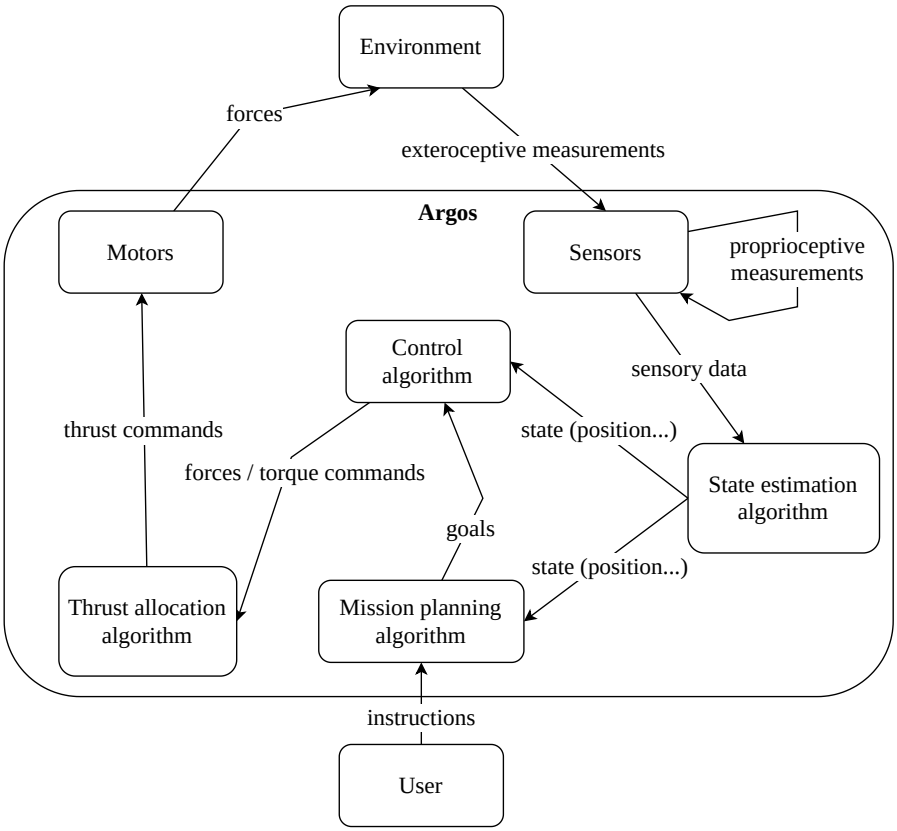


Figure 1.6: Software architecture of Argos, an autonomous ROV developed at Forssea Robotics (https://forssea-robotics.fr/index.php/products/rovs)

Figure 1.7: Argos manoeuvring to enter its garage (Courtesy of Forssea Robotics/Balao)

which is a discrete machine: its programs are paced by a clock, such that it issues commands and reads sensors data at a specific pace. Therefore, an autonomous ROV (and more generally any robot) can be modelled as a *hybrid system*, with continuous- and discrete-time processes interacting together. Thus, our goal is to prove feasibility of a docking mission for a specific robot (or hybrid system). In other words, we want to find a method capable of validating the engineering choices made for a particular robot. Again, we believe that such a method can be of interest, especially for industrial applications, where proofs of correct functioning are usually required to allow the large scale deployment of a system. Indeed, knowing that a docking mission will always succeed if the robot starts from a specific area, or that it might fail if it is not equipped with a particular sensor will be of interest to the designer and the users of such a robot.

It is worth noting that similar methods exist in different domains. Special programming languages do exist to validate software for critical systems [20], rigorously simulating and verifying hybrid systems [3], [12], [118]. However, these tools do not fully meet our requirements, because they concentrate on the software part of a system, they focus primarily on simulating the system, or they are mainly intended for linear systems verification. In our case, we want to deal with non-linear uncertain hybrid systems and prove *a priori* that a robot can dock into its garage if initialised in a given area.

In this thesis, we explored two different approaches to solve this problem. We refer the reader to Chapter 2 for a more in-depth introduction of the mathematical tools and concepts mentioned below.

1.2.2.2 *Reachability analysis*

Checking that a system can reach a specific area is known as *reachability analysis*. Multiple methods exist to verify such a property (see [1] for an extensive list and presentation of related research subjects). The most intuitive one, especially in an industrial context, consists in modelling the system using a mathematical formalism and simulating the outcomes resulting from different initial conditions: tools such as the ubiquitous Matlab/Simulink, or the open source Gazebo simulation software are often used in robotics industries. Such a method often leads to simulating as many outcomes as possible to obtain a rough idea of the system's behaviour with various initial conditions. While easy to understand and implement, this method lacks the mathematical guarantees we seek for our approach. To meet this requirement, methods falling in the field of *symbolic reachability analysis* can be used [2], [36], [38], [69], [95], [97]. Their working principle consists in enclosing all the possible states taken by a system inside a mathematical object, and studying the latter's evolution. Then, the

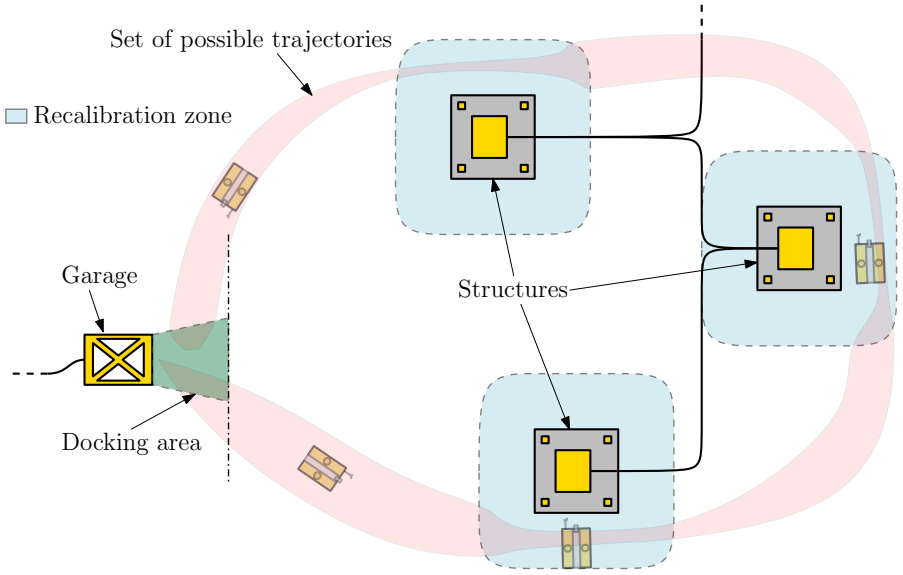


Figure 1.8: Reachability analysis of a robot leaving from its garage, equipped with a specific set of sensors and controlled by a specific controller. When entering the blue areas, the robot is able to recalibrate its estimated position.

proof of reachability comes down to checking whether it is contained inside the desired specific area. Similarly, proving that a system will not enter a specific zone can be done using the same tools. The reader may refer to Chapter 4 to have more details about this approach.

1.2.2.3 *Stability analysis*

We will take the following image to introduce this approach: imagine that the robot is a marble, evolving on a cloth presenting hills and valleys. Assume also that the robot has to reach one of the latter. Proving that the robot will end up in the desired valley comes down to proving that it is initialised in its catchment; or that it has enough speed and the right heading to climb the hills separating it from the right valley. This approach falls into the field of *stability analysis*. Again, various methods are available to study the stability of a system [41]. Their working principles differ, ranging from eigenvalues evaluation to the use of Lyapunov's theorems. Proving stability for an uncertain system has also been studied [49], [101], as well as stability of hybrid systems [14], [35]. We refer the reader to Chapter 3 for more details about this approach.

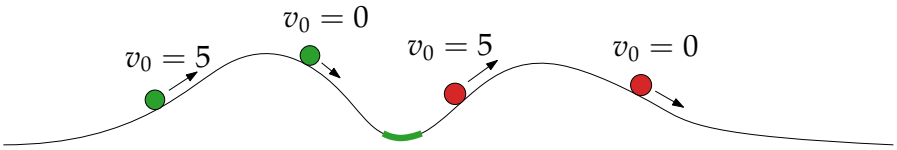


Figure 1.9: Stability analysis of a marble on a cloth: the green marbles will end up in the green valley, because of their initial position or their initial velocity, to the contrary of the red marbles.

1.3 CONTRIBUTIONS

Aside from the two approaches we studied to prove feasibility of a docking mission, we developed various tools that can have a broader use than the one for which they were initially intended. We listed the main ones below.

- Comprehensive derivation of Lohner's algorithm, and an example of implementation (see Section 2.4.5);
- Algorithm to compute the centred form of an iterated function, to allow for tighter image set enclosures (see Section 3.4.2);
- Stability contractor, a formalism allowing to easily prove stability of a system (see Section 3.4.1);
- Lohner contractor for tubes, to obtain tighter enclosures for trajectories, and its implementation in the Tubex library (see Section 4.3.2);
- Comprehensive examples using the CAPD library, to broaden its usage among the interval robotics community.

2

MATHEMATICAL TOOLS AND CONCEPTS

TABLE OF CONTENTS

2.1 INTRODUCTION

In this chapter, we will present the mathematical tools and the concepts used later in this thesis.

We recall that this work's goal is to propose a method to *prove* feasibility of a docking mission between a *robot* and an underwater asset.

Therefore, we start by introducing the different ways of modelling *dynamical systems*, and robots more specifically. Doing so will provide

us with a mathematical tool to predict a robot's behaviour in time, given an initial state. This subject is tackled in Section 2.2.

Now, as good as a model can be, it will always approximate the real-life system, for reasons ranging from lack of computational power to unknown effects acting on the robot, to unpredictable events occurring during an operation. Since we seek mathematical proof of feasibility of a mission, we will need to use specific tools to consider those uncertainties while predicting the robot's behaviour. In this thesis, the tool we chose to deal with the modelling of errors and uncertainties is *interval analysis*, which we introduce in Section 2.3.

We then introduce different methods to model dynamical systems' evolution using interval analysis in Section 2.4. Thanks to the latter, these methods perform *guaranteed integration* of a system's model, which can then be interpreted to ensure its properties.

2.2 MODELLING ROBOTS

Mobile robots are unmanned vehicles, sometimes autonomous, which evolve on land, in the air, on or under water, or even in space. These robots can usually sense their environment and act on it via sensors, actuators and an internal computer. Therefore, their behaviour is dictated by the laws of physics and the internal logic of the embedded computer [32], [48], [111].

2.2.1 *Dynamical systems*

Mathematically speaking, mobile robots can be modelled as dynamical systems, i. e. a set of mathematical equations describing the system's evolution in time. A general definition for a dynamical system can be found in [37, Chapter 1]:

Definition 2.1. A dynamical system is a function $\phi : T \times \mathcal{S} \to \mathcal{S}$ such that

If $T = \mathbb{N}$ or $T = \mathbb{R}^+$, it is common to talk about semi dynamical systems.

1. T is one of the following: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^+$. $t \in T$ is the *evolution parameter* of the system and T its *time set*;
2. $\mathcal{S}$ is a non-empty set, the *state space* of the system containing its *state* $\mathbf{x}$;
3. $\phi(0, .)$ is the identity function: for any $\mathbf{x} \in \mathcal{S}$, $\phi(0, \mathbf{x}) = \mathbf{x}$;
4. $\phi(t, \phi(s, \mathbf{x})) = \phi(t + s, \mathbf{x})$ for any $t, s \in T$ and for any $\mathbf{x} \in \mathcal{S}$.

Remark 2.1. The Item 3 imposes that the system's state cannot change over a null duration, i. e. it cannot instantaneously teleport itself to

another point in its state space. The Item 4 imposes that the future system's state depends on the current one.

Let us introduce a few notations and terms:

- If we take $\mathbf{x} = \mathbf{x}_0$ as constant, then it is the *initial state* of the system. $\phi_{\mathbf{x}_0}(t)$ is the *flow* of the system passing through $\mathbf{x}_0$;
- If we take the duration t as constant, then $\phi_t(\mathbf{x})$ is called the *state transition of duration t*, *t-transition* or *t-advance* of the system;
- If $T = \mathbb{R}$ or $\mathbb{R}^+$, the system is called *continuous-time* (see Section 2.2.2)
- If $T = \mathbb{N}$ or $\mathbb{Z}$, the system is said to be *discrete* (see Section 2.2.3).

Remark 2.2. In the rest of this thesis, we will assume that $T = \mathbb{R}$ or $\mathbb{Z}$, although t will be positive most of the time. We will also assume that $\mathcal{S} = \mathbb{R}^n$, where $n \in \mathbb{N}^*$.

2.2.2 *Continuous-time dynamical systems*

2.2.2.1 *Definition*

Continuous-time dynamical systems are used to model phenomenons that evolve continuously over time. These are widely found in real-life problems' models, from moving vehicles to biological and chemical processes. Definition 2.1 can be adapted to continuous dynamical systems [41, Chapter 7], [92, Chapter 3]:

Definition 2.2. A continuous-time dynamical system is a function $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ such that

1. $\phi(0, .)$ is the identity function: for any $\mathbf{x} \in \mathbb{R}^n$, $\phi(0, \mathbf{x}) = \mathbf{x}$
2. $\phi(t, \phi(s, \mathbf{x})) = \phi(t + s, \mathbf{x})$ for any $t, s \in \mathbb{R}$ and for any $\mathbf{x} \in \mathbb{R}^n$.

2.2.2.2 *Relation between dynamical systems and ordinary differential equations*

Differential equations (ordinary or partial) are widely used to model continuous phenomenons: they are used for systems ranging from atmospheric phenomenons [41, Chapter 14], [76], [123] to electrical oscillators' behaviour [41, Chapter 12], [127], biological processes [41, Chapter 11], [47], physical & astronomical phenomenons [41, Chapter 13], [57], [132], [133], and vehicles' and robots' trajectories [22, Chapter 4], [32, Chapter 7], [48, Chapter 1], [111, Part A & F]. To model

the latter, one usually uses classical mechanics and usually obtains an Ordinary Differential Equation (ODE) such as 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 $\mathbf{x}(t) \in \mathbb{R}^n$ the robot's state. This equation fully describes the system's behaviour, and only depends on its current state: it is said to be *autonomous*. Sometimes, the function $\mathbf{f}$ can also depend on an input $\mathbf{u}(t)$, which can be used to take external commands into account.

Autonomous systems are closely related to dynamical systems, in the sense that Equation (2.1) has a unique solution $\phi_t(\mathbf{x}_0)$ passing through a point $\mathbf{x}_0 \in \mathbb{R}^n$ at a given time $t = 0$. This solution actually corresponds to the flow of the associated dynamical system initialised at $\mathbf{x}_0 \in \mathbb{R}^n$ and $t = 0$.

This is a consequence of the existence and uniqueness theorem [41, Chapter 7], [59, Chapter 3].

Remark 2.3. It is rarely possible to find an analytical expression for $\phi(t, \mathbf{x})$, except in particular cases, e. g. when $\mathbf{f}$ is linear with respect to $\mathbf{x}$. However, using numerical methods, it is possible to evaluate ϕ at a specific time for a specific initial state [41, Chapter 7].

Also, note that for some ODEs, some solutions might only be defined locally in time, i. e. might blow up to infinity in a finite time. Therefore, they may be used to define *local dynamical systems* only.

Definition 2.3. Solving an Initial Value Problem (IVP) consists in finding the trajectory of a system described by

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \tag{2.2}$$

over a time interval $[t_0, t_f]$.

IVPs are very practical since their solution is actually a prediction of the system's state evolution over time. In the rest of this thesis, most of our work somehow consists in solving IVPs.

Geometrically, in the state space of the system, Equation (2.1) can be interpreted as a vector field, and its flow as a field line passing through a specific point, or equivalently a trajectory of the system.

Example 2.1. Consider for example the Van der Pol oscillator [127], described by the following equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ (1 - x_1^2)\, x_2 - x_1 \end{pmatrix} \tag{2.3}$$

The state space of the system is $\mathbb{R}^2$ and its time set is $\mathbb{R}$. In Figure 2.1, we have represented the concepts presented earlier: vector field, flow and t-transition.

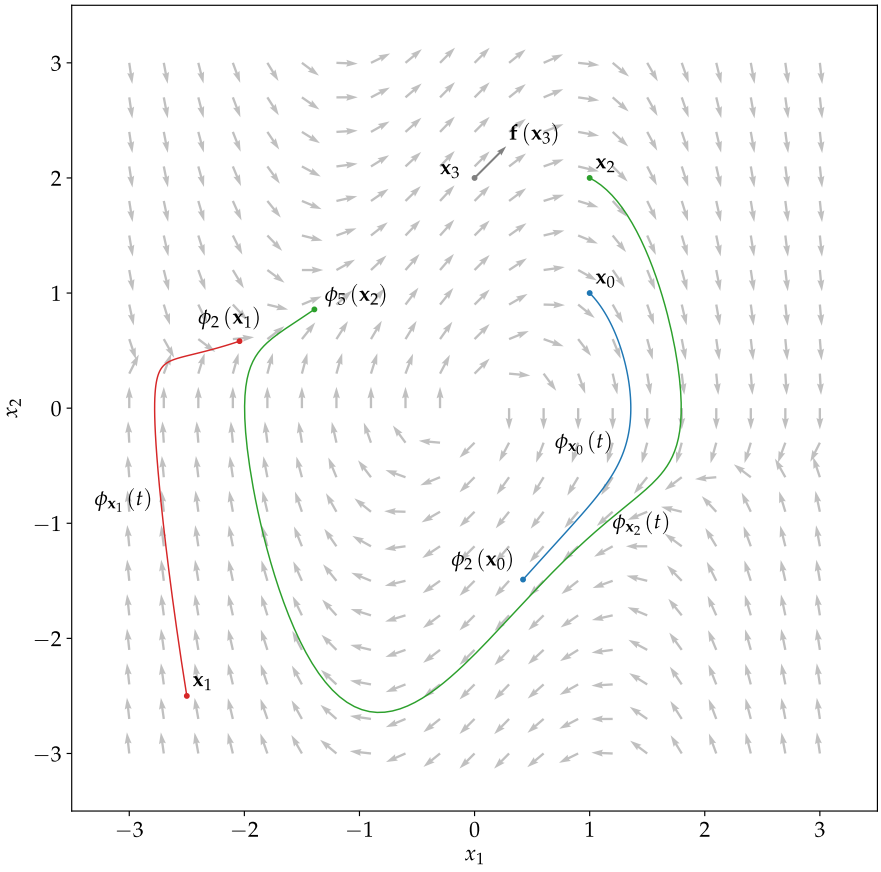


Figure 2.1: Illustration of the Van der Pol oscillator

2.2.2.3 *Derivatives of the flow*

Depending on the application, one might need to compute the derivatives of ϕ with respect to t or $\mathbf{x}$. The following theorems, derived in [41, Chapter 7], are of importance when doing so.

2.2.2.3.1 ABOUT THE EXISTENCE OF THE SPACE AND TIME DERIVATIVES

Theorem 2.1. *Consider the system* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *where* $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$. *Then* $\phi(t, \mathbf{x})$ *is* $\mathcal{C}^1$, *i.e.* $\frac{\partial \phi}{\partial t}$ *and* $\frac{\partial \phi}{\partial \mathbf{x}}$ *exist and are continuous in* t *and* $\mathbf{x}$.

...provided that $\mathbf{x}(t)$ *is defined.*

In particular, considering a point $\mathbf{x}_0 \in \mathbb{R}^n$ the following equations stand

$$\frac{\partial \phi}{\partial t}(t, \mathbf{x}_0) = \mathbf{f}(\phi(t, \mathbf{x}_0)) \tag{2.4}$$

$$\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0) = \mathbf{J}_{\phi_t}(\mathbf{x}_0) \tag{2.5}$$

where $\mathbf{J}_{\phi_t}$ is the Jacobian of $\phi_t(\mathbf{x})$.

Thus, computing the time-derivative of ϕ comes down to evaluating the function $\mathbf{f}(\phi(t, \mathbf{x}_0))$. However, because finding an analytical expression for $\phi(t, \mathbf{x})$ is not always possible, another approach than symbolic differentiation is required to calculate $\frac{\partial \phi}{\partial \mathbf{x}}$.

2.2.2.3.2 ABOUT THE VARIATIONAL EQUATION

Definition 2.4 (Variational equation)**.** Consider a system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$. Let $\mathbf{x}(t)$ be a trajectory of the system defined on the time interval $[t_0, t_f]$, such that $\mathbf{x}(t_0) = \mathbf{x}_0$. Then the *variational equation* along $\mathbf{x}(t)$ is given by

$$\dot{\mathbf{U}}(t) = \mathbf{A}(t) \cdot \mathbf{U}(t) \tag{2.6}$$

where $\mathbf{A}(t) = \mathbf{J}_{\mathbf{f}}(\mathbf{x}(t))$ is the Jacobian of $\mathbf{f}$ evaluated at $\mathbf{x}(t)$ and $\mathbf{U}(t)$ is a matrix.

Remark 2.4. The initial condition for $\mathbf{U}(t)$ is often chosen to be the identity matrix. This choice will be explained in the next paragraph.

Equation (2.6) is called the *variational equation* [41, Chapter 7] (or *sensitivity equation* [59, Chapter 3]) along the solution $\mathbf{x}(t)$. This equation has a solution for every initial condition $\mathbf{U}_0$.

Note that this equation is often given in a vector form instead of a matrix one in textbooks

$$\dot{\mathbf{u}}(t) = \mathbf{A}(t) \cdot \mathbf{u}(t) \tag{2.7}$$

Using this notation, it is easier to understand the denomination *sensitivity equation*: this equation allows studying the sensitivity of the trajectory $\mathbf{x}(t)$ to the initial condition $\mathbf{x}_0$. The next example illustrates this property.

Example 2.2. Consider the Van der Pol oscillator described by Equation (2.3). Consider a trajectory of this system $\mathbf{x}(t)$ such that $\mathbf{x}(0) = \mathbf{x}_0$ and $t \in [t_0, t_f]$. Consider another trajectory $\mathbf{y}(t)$ of the system initialised near the first one at $\mathbf{x}_0 + \mathbf{u}_0$, $\mathbf{u}_0$ being a small vector. Let us define the vector

$$\mathbf{u}(t) = \mathbf{y}(t) - \mathbf{x}(t)$$

I. e. for every $t \in [t_0, t_f]$, $\mathbf{u}(t)$ corresponds to the error vector between the trajectories $\mathbf{x}(t)$ and $\mathbf{y}(t)$.

In Figure 2.2, we plotted the trajectories $\mathbf{x}(t)$ and $\mathbf{y}(t)$, as well as the vector $\mathbf{u}(t)$ at arbitrary moments in time. To illustrate the matrix form of the variational equation, we also plotted the trajectories of the system initialised in $\mathbf{c}_1$, $\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{u}_1$ and $\mathbf{y}_2 = \mathbf{x}_1 + \mathbf{u}_2$. The

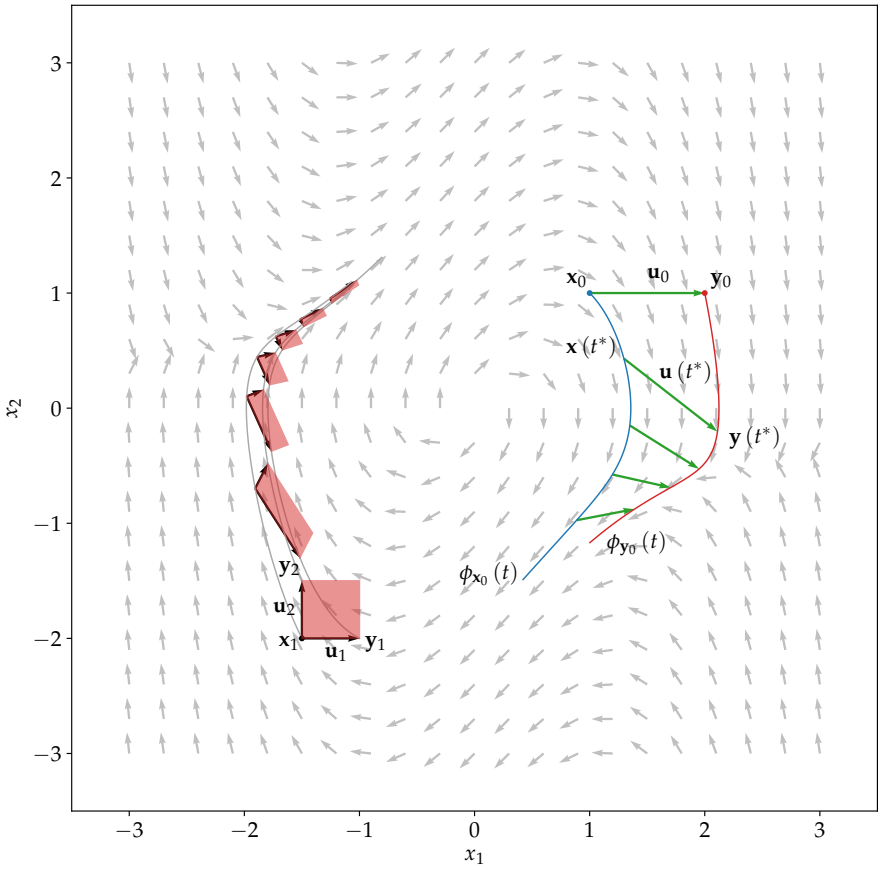


Figure 2.2: Impact of the initial state on the trajectory - variational equation

red parallelepiped helps understand how sensible the system is with respect to the state vector's different components.

Now, let us differentiate $\mathbf{u}(t)$ with respect to time

$$\begin{aligned}\frac{d\mathbf{u}}{dt}(t) &= \frac{d\mathbf{y}}{dt}(t) - \frac{d\mathbf{x}}{dt}(t)\\ &= \mathbf{f}(\mathbf{y}(t)) - \mathbf{f}(\mathbf{x}(t))\\ &= \mathbf{f}(\mathbf{x}(t) + \mathbf{u}(t)) - \mathbf{f}(\mathbf{x}(t))\\ &= \mathbf{f}(\mathbf{x}(t)) + \frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{x}(t)) \cdot \mathbf{u}(t) + o(\mathbf{u}(t)) - \mathbf{f}(\mathbf{x}(t))\\ &\simeq \mathbf{J}_{\mathbf{f}}(\mathbf{x}(t)) \cdot \mathbf{u}(t)\end{aligned}$$

where $\mathbf{J}_{\mathbf{f}}$ is the Jacobian of $\mathbf{f}$. Let us define $\mathbf{A}(t) = \mathbf{J}_{\mathbf{f}}(\mathbf{x}(t))$, and we obtain Equation (2.7).

2.2.2.3.3 ABOUT THE LINK BETWEEN SPACE-DERIVATIVE AND VARIATIONAL EQUATION

We will now see that the variational equation can be used to compute the space-derivative of a trajectory.

Consider a system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$ for $t \in [t_0, t_f]$. Let $\mathbf{x}(t_0) = \mathbf{x}_0$ be an initial state in $\mathbb{R}^n$. Then for every $t \in [t_0, t_f]$,

$$\phi(t, \mathbf{x}_0) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{f}(\phi(s, \mathbf{x}_0))\, ds$$

Differentiating this equation with respect to $\mathbf{x}$ yields

$$\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0) = \int_{t_0}^{t} \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\phi(s, \mathbf{x}_0)) \cdot \frac{\partial \phi}{\partial \mathbf{x}}(s, \mathbf{x}_0)\, ds$$

Differentiating the last equation with respect to time yields

$$\frac{d}{dt}\left(\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0)\right) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\phi(t, \mathbf{x}_0)) \cdot \frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0)$$

Finally, let us define $\mathbf{u}(t) = \frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0)$ and $\mathbf{A}(t) = \mathbf{J}_{\mathbf{f}}(\phi(t, \mathbf{x}_0))$, and we obtain Equation (2.6)

$$\dot{\mathbf{u}}(t) = \mathbf{A}(t) \cdot \mathbf{u}(t)$$

The matrix form of this last equation can be obtained by choosing two different initial conditions and by concatenating the resulting vector variational equations.

According to [41, Chapter 7], one can compute $\frac{\partial \phi}{\partial \mathbf{x}}$ using the following theorem.

Theorem 2.2. *Consider a system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$. Let $\mathbf{x}(t)$ be a solution defined for $t \in [t_0, t_f]$ such that $\mathbf{x}(t_0) = \mathbf{x}_0$. Let $\mathbf{U}(t)$ be the solution of the variational equation along $\mathbf{x}(t)$ such that $\mathbf{U}(t_0) = \mathbf{U}_0$. Then*

$$\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0) \cdot \mathbf{U}_0 = \mathbf{U}(t) \tag{2.8}$$

In other words, computing $\frac{\partial \phi}{\partial \mathbf{x}}$ for a given initial state $\mathbf{x}_0$ at a certain time t simply comes down to integrating the associated variational equation for a duration t, in particular if we choose $\mathbf{U}_0 = \mathbf{I}_n$. We will now illustrate this theorem with a few examples.

Example 2.3. Consider the system described by the following differential equation

The system is intentionally chosen linear to ease calculations.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \mathbf{x} = \mathbf{A} \cdot \mathbf{x}$$

On the one hand, one can easily compute the trajectory of the system initialised at $\mathbf{x}(0) = \mathbf{x}_0$

$$\phi(t, \mathbf{x}_0) = e^{\mathbf{A}t} \cdot \mathbf{x}_0$$

Then, differentiating ϕ with respect to $\mathbf{x}$ yields

$$\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0) = e^{\mathbf{A}t}$$

On the other hand, let us define the variational equation along the solution $\mathbf{x}(t)$, $\mathbf{x}(0) = \mathbf{x}_0$

$$\dot{\mathbf{u}}(t) = \mathbf{A}(t) \cdot \mathbf{u}(t)$$

The solution of the latter equation, when $\mathbf{u}(0) = \mathbf{u}_0$ is

$$\mathbf{u}(t) = e^{\mathbf{A}t} \cdot \mathbf{u}_0$$

As Theorem 2.2 states, we have

$$\frac{\partial \phi}{\partial \mathbf{x}}(t, \mathbf{x}_0) \cdot \mathbf{u}_0 = e^{\mathbf{A}t} \cdot \mathbf{u}_0 = \mathbf{u}(t)$$

Example 2.4. Consider a simple pendulum, which state vector $\mathbf{x}$ is composed of its angle θ and its angular speed ω, described by the following differential equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{cases} \omega \\ -\sin(\theta) - \omega \end{cases} \tag{2.9}$$

Let $\mathbf{x}(t)$ be a solution of Equation (2.9) with $\mathbf{x}(0) = \mathbf{x}_0$, and $\mathbf{y}(t)$ be a solution of Equation (2.9) with $\mathbf{y}(0) = \mathbf{x}_0 + \mathbf{u}_0$ where $\mathbf{u}_0$ is a small vector of $\mathbb{R}^2$. It is not possible to find a analytic expression for $\mathbf{x}(t)$, so the goal of this example will not be to illustrate Theorem 2.2, but one of its consequences. We will show graphically that we can approximate a trajectory $\mathbf{y}(t)$ initialised near $\mathbf{x}_0$ using the variational equation along $\mathbf{x}(t)$.

But first, let us express the latter

$$\dot{\mathbf{u}}(t) = \mathbf{J}_{\mathbf{f}}(\mathbf{x}(t)) \cdot \mathbf{u}(t) = \begin{bmatrix} 0 & 1 \\ -\cos(\theta) & -1 \end{bmatrix} \cdot \mathbf{u}(t) \tag{2.10}$$

Let $\mathbf{u}(t)$ be the solution of Equation (2.10) with $\mathbf{u}(0) = \mathbf{u}_0$. Equations (2.9) and (2.10) can be concatenated into the following system

$$\dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}) = \begin{pmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{u_1} \\ \dot{u_2} \end{pmatrix} = \begin{pmatrix} \omega \\ -\sin(\theta) - \omega \\ u_2 \\ -\cos(\theta) u_1 - u_2 \end{pmatrix} \tag{2.11}$$

Note that it is necessary to rearrange the system as such since $\mathbf{J}_{\mathbf{f}}$ depends on $\mathbf{x}(t)$: both Equations (2.9) and (2.10) must be integrated jointly.

Now, we can numerically integrate Equation (2.9) with $\mathbf{x}_0$, Equation (2.9) with $\mathbf{y}_0 = \mathbf{x}_0 + \mathbf{u}_0$ and Equation (2.11) with $\mathbf{z}(0) = (\mathbf{x}_0, \mathbf{u}_0)^{\mathrm{T}}$.

Define the trajectory $\hat{\mathbf{y}}(t) = \mathbf{x}(t) + \mathbf{u}(t)$. Figure 2.3 shows that both $\mathbf{y}(t)$ and $\hat{\mathbf{y}}(t)$ evolve closely for a certain time, before diverging.

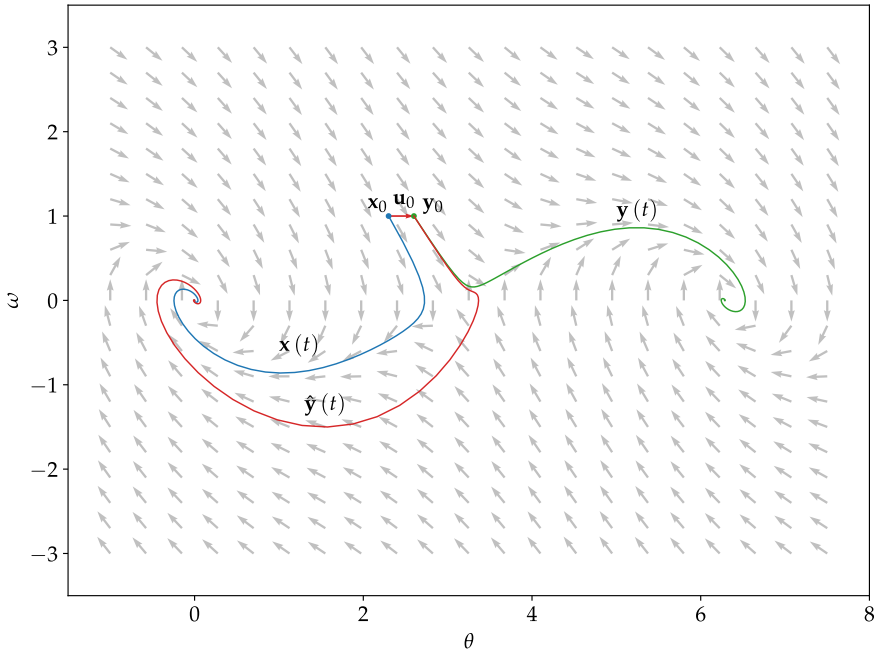


Figure 2.3: Approximation of a trajectory using the variational equation

2.2.2.4 *Conclusion*

To sum up, we recalled a definition of a continuous dynamical system and presented a few related concepts. More will be presented in Section 3.2. We recall the importance of such a mathematical tool, as it is the most used to describe the evolution over time of a mechanical system, and particularly of vehicles and robots.

2.2.3 *Discrete dynamical systems*

2.2.3.1 *Definition*

Discrete dynamical systems are used to model phenomenons occurring punctually, at specific moments in time. Following Definition 2.1, let us propose the following definition of a discrete dynamical system:

This actually defines a discrete semi-dynamical system. For the sake of simplicity, we will drop the "semi" in the rest of this thesis.

Definition 2.5. A discrete dynamical system is a function $\phi : \mathbb{N} \times \mathbb{R}^n \to \mathbb{R}^n$ such that

1. $\phi(0, .)$ is the identity function: for any $\mathbf{x} \in \mathbb{R}^n$, $\phi(0, \mathbf{x}) = \mathbf{x}$
2. $\phi(p, \phi(q, \mathbf{x})) = \phi(p + q, \mathbf{x})$ for any $p, q \in \mathbb{N}$ and for any $\mathbf{x} \in \mathbb{R}^n$.

Remark 2.5. In the rest of this thesis, when dealing with discrete-time systems, we will drop the first argument of ϕ for the sake of

clarity. Then, we will write $\phi(1, \mathbf{x}) = \phi(\mathbf{x})$ and $\phi(k, \mathbf{x}) = \phi^k(\mathbf{x}) = \underbrace{(\phi \circ \cdots \circ \phi)}_{k}(\mathbf{x})$.

Note that k does not necessarily correspond to a number of time units, but merely to the number of mapping by ϕ.

2.2.3.2 *Relation between discrete systems and mathematical sequences*

Consider a function $\phi : \mathbb{N} \times \mathbb{R}^n \to R^n$, and a vector $\mathbf{x}_0 \in \mathbb{R}^n$ representing the initial state of a discrete system. Then the successive mapping of $\mathbf{x}_0$ by ϕ forms a sequence

$$\left(\mathbf{x}_0, \phi(\mathbf{x}_0), \phi \circ \phi(\mathbf{x}_0), \ldots, \phi^k(\mathbf{x}_0)\right) = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$$

where $k \in \mathbb{N}$. Therefore, we will often use an equation defining a mathematical sequence to describe a discrete-time system

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \tag{2.12}$$

2.2.3.3 *Examples of discrete systems*

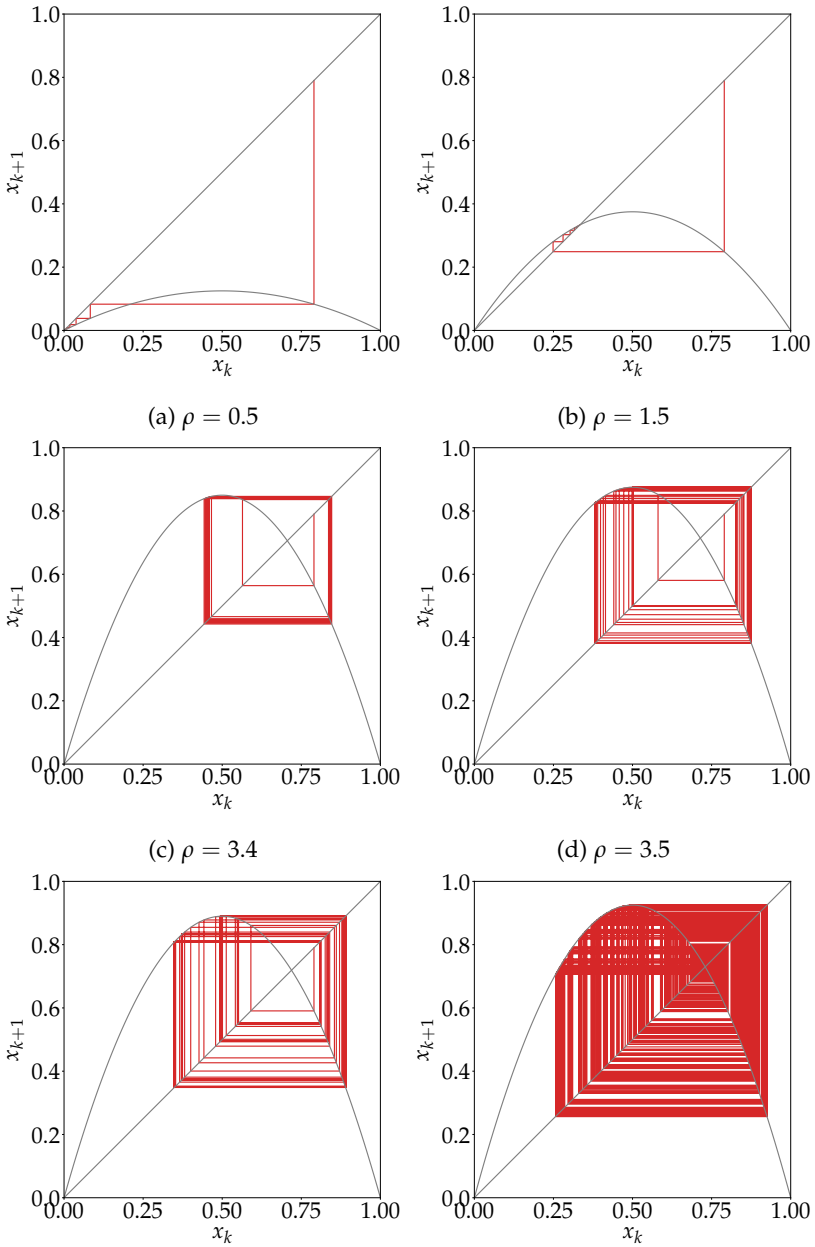
Example 2.5. A classic example of discrete system is the logistic map. This one-dimensional system is described by Equation (2.13), where $\rho \in [0, 4]$ is a parameter and $x_0 \in [0, 1]$. This very simple system in appearance can actually display highly complex behaviours and becomes chaotic for $\rho \geq 3.57$.

$$x_{k+1} = \rho \cdot x_k \cdot (1 - x_k) \tag{2.13}$$

In Figure 2.4, we represented the various behaviours of the system for different values of ρ. In each case, we chose $x_0 = 0.79$, and we computed x_k up to $k = 500$. To build the trajectory in the plane (x_k, x_{k+1}), we start by choosing x_0 and place it on the line $x_{k+1} = x_k$, i. e. at the point (x_0, x_0). Then, we join the parabola, which corresponds to the logistic map, at the point (x_0, x_1). We then project this point on the line, and start the process repeatedly to find the sequence (x_k).

While Figures 2.4a to 2.4e display a trajectory converging towards a point or oscillations between specific points, Figure 2.4f is a typical example of chaotic behaviour: a slight change in the initial value of x_0 will result in a completely different trajectory. More insight about the logistic map will be given in Section 3.2.2.

Example 2.6. Discrete dynamical systems are often used in robotics to model a continuous system that has been discretised over time (in that case, the index k can actually refer to a number of time units), for example when implementing a Kalman filter [48, Chapter 7], [121,



(a) $\rho = 0.5$ (b) $\rho = 1.5$

(c) $\rho = 3.4$ (d) $\rho = 3.5$

(e) $\rho = 3.56$ (f) $\rho = 3.7$

Figure 2.4: Trajectory (in red) of the logistic map in the plane (x_k, x_{k+1})

Chapter 3] or a numerical integration method. Consider the differential equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. Let $\mathbf{x}_0$ be the initial state of the system. Then, a simple Euler integration method can be seen as a discrete system:

$$\mathbf{x}_{n+1} = \phi(\mathbf{x}_n)$$

where

$$\phi(\mathbf{x}_n) = \mathbf{x}_n + h\mathbf{f}(\mathbf{x}_n)$$

2.2.4 *Hybrid dynamical systems*

Hybrid dynamical systems are used to model the interactions between a continuous-time and a discrete-time system [70]. These systems are typically found in robotics: a mobile robot is a physical system evolving in the real world (hence the continuous part of the model) being driven by a computer (which is an inherent discrete system).

2.2.4.1 *Definition*

Mathematically speaking, a hybrid system is often defined as a sextuple [54], [112] or a septuple [15], [34], [35], depending on the necessity of a control model in the hybrid system.

Definition 2.6. A hybrid system is a sextuple $\mathcal{H} = (\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{F}, \mathcal{G}, \mathcal{R})$ where:

1. $\mathcal{Q} = \{q_0, \dots, q_m\}$ is the countable set of discrete states of $\mathcal{H}$, with cardinality $m+1$;
2. $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ is the set of *edges* (or *transitions*) between discrete states;
3. $\mathcal{D} = \left\{D_{q_i}, q_i \in \mathcal{Q}\right\}$ is the collection of *domains* of $\mathcal{H}$, where $\forall q_i \in \mathcal{Q}$, D_{q_i} is a non-empty subset of $\mathbb{R}^n$;
4. $\mathcal{F} = \left\{\mathbf{f}_{q_i}, q_i \in \mathcal{Q}\right\}$ is the collection of *vector fields*, where for every $q_i \in \mathcal{Q}$, $\mathbf{f}_{q_i} \in \mathcal{C}^1\left(D_{q_i}\right)$ and its associated dynamical system is denoted $\phi_{q_i}(t, \mathbf{x})$;
5. $\mathcal{G} = \{G_e, e \in \mathcal{E}\}$ is the collection of *guards*, each guard delimiting a boundary between two sets D_{q_i} and D_{q_j}: $\forall e = \left(q_i, q_j\right) \in \mathcal{E}$, $G_e \subset D_{q_i}$;
6. $\mathcal{R} = \{R_e, e \in \mathcal{E}\}$ is the collection of *resets*, such that for each $e = \left(q_i, q_j\right) \in \mathcal{E}$, $R_e : G_e \to 2^{D_{q_j}}$, where $2^{D_{q_j}}$ denotes the powerset of D_{q_j}. In the rest of this thesis, we will simply consider R_e as a map $R_e : G_e \to D_{q_j}$.

The state of a hybrid system is therefore composed of a discrete variable $q_i \in \mathcal{Q}$ and of a continuous variable $\mathbf{x} \in \mathbb{R}^n$. Its evolution through time is called *execution* and is represented in Figure 2.5: the system is in a given state $(q_i, \mathbf{x})$. The continuous variable $\mathbf{x}$ evolves through time in the set D_{q_i} according to the vector field defined by $\mathbf{f}_{q_i}$, until reaching the guard G_e, $e = \left(q_i, q_j\right)$. When the guard is reached, the associated reset R_e is triggered and resets the value of $\mathbf{x}$. Then, the discrete variable q_i switches to q_j, and the execution keeps going.

Remark 2.6. 1. One can notice that the system evolves exactly like a continuous dynamical system in between the discrete transitions.

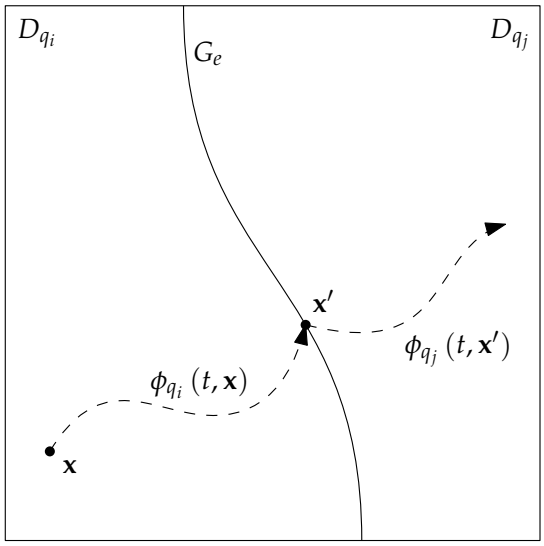


Figure 2.5: Illustration of the execution of a hybrid system

2. A hybrid system may only have one discrete state q (see Section 2.2.4.2).

2.2.4.2 *Examples of hybrid systems*

Let us give some examples to illustrate the definition. We will start by a prevalent one, the bouncing ball, and then apply the definition to a simple robot driven by a discrete controller.

Example 2.7 (Bouncing ball). Imagine a ball made out of rubber, free-falling and hitting the ground. After impact, the ball will bounce back up, although the ground has absorbed a part of the energy. This simple phenomenon can be modelled as a hybrid system.

Let z be the altitude of the ball above the ground, and v its vertical speed. The continuous part of the state is $\mathbf{x} = (z, v) \in \mathbb{R}^2$. There is only one discrete state q, since the ball is always bouncing in the same subset of $\mathbb{R}^2$,

$$D_q = \left\{ \mathbf{x} \in \mathbb{R}^2 \,\middle|\, z \geq 0 \right\}$$

The ball's behaviour is described by the differential equation

$$\dot{\mathbf{x}} = \mathbf{f}_q(\mathbf{x}) = (v, -mg)$$

where m is the ball's mass and g the acceleration of gravity
The system has a single guard

$$G_{(q,q)} = \left\{ \mathbf{x} \in \mathbb{R}^2 \,\middle|\, z = 0, v \leq 0 \right\}$$

which is crossed whenever the ball touches the ground; and a single reset function

$$R_{(q,q)}(\mathbf{x}) = (0, -cv)$$

where c is a constant modelling the energy lost during the impact.

Figure 2.6 illustrates the execution of this model.

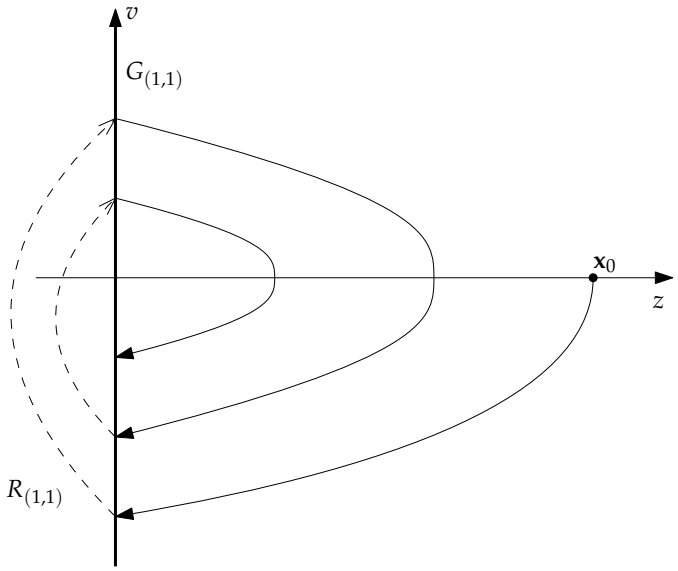


Figure 2.6: Execution of the model of a bouncing ball, in the phase plane (z, v)

Example 2.8 (Robot control). Consider a robot modelled as a Dubin's car, equipped with all the sensors required to perform a full state estimation at a given frequency f (let $h = 1/f$ be the associated period).

The continuous part of the robot's state is composed of its position and heading, but also of a time variable τ and a desired angular velocity ω_d: $\mathbf{x} = (x, y, \theta, \tau, \omega_d) \in \mathbb{R}^5$. While τ increases linearly with time (it is the embedded timer of the system), ω_d is a constant updated at the frequency f.

Adding τ to the state vector allows defining a guard with respect to time, which in turn allows resetting the desired heading θ_d at a given frequency. Adding ω_d makes the differential equation autonomous between two command updates.

There is only one possible discrete state q, since the robot evolves in a single subset of $\mathbb{R}^n$

$$D_q = \left\{ \mathbf{x} \in \mathbb{R}^5 \,\middle|\, \tau \leq h \right\}$$

The robot's behaviour is driven by the following differential equation

$$\dot{\mathbf{x}} = \mathbf{f}_q(\mathbf{x}) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\tau} \\ \dot{\omega}_d \end{pmatrix} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ \omega_d \\ 1 \\ 0 \end{pmatrix}$$

The single guard of the system is defined by

$$G_{(q,q)} = \left\{ \mathbf{x} \in \mathbb{R}^5 \,\middle|\, \tau = h \right\}$$

And crossing the latter triggers the reset function

$$R_{(q,q)}(\mathbf{x}) = (x, y, \theta, 0, \sin(\theta_d - \theta))$$

where θ_d is the desired heading for the robot.

Therefore, the robot moves according to a command $\omega_d = \sin(\theta_d - \theta)$ issued at a frequency f, i. e. when the time variable τ reaches h. Figure 2.7 illustrates this system's execution with $f = 0.33$ Hz and $\theta_d = \pi/4$, and Figure 2.8 represents the values taken by the state vector's components over time.

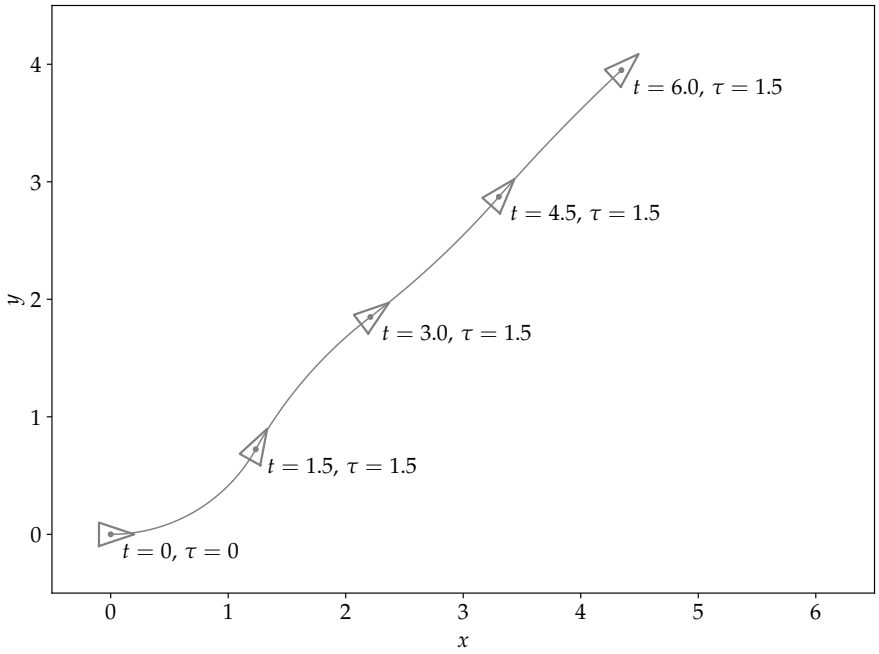


Figure 2.7: Execution of the robot in the plane (x, y)

2.3 INTERVAL ANALYSIS

It is rarely possible to build a perfect model for a robot: the latter's parameters are possibly not entirely known, additional forces can act on it and modify its behaviour, sensory data is often noisy... Therefore if one wants to predict a robot's behaviour, these uncertainties must be taken into account to reach a plausible prediction. In this thesis, we chose to use interval analysis to represent uncertainties, whether from model, measurement or control data. Therefore, each uncertain quantity will be described as an interval containing the exact unknown value, thus allowing to perform guaranteed, but pessimistic, computations. The goal of this section is to introduce this field.

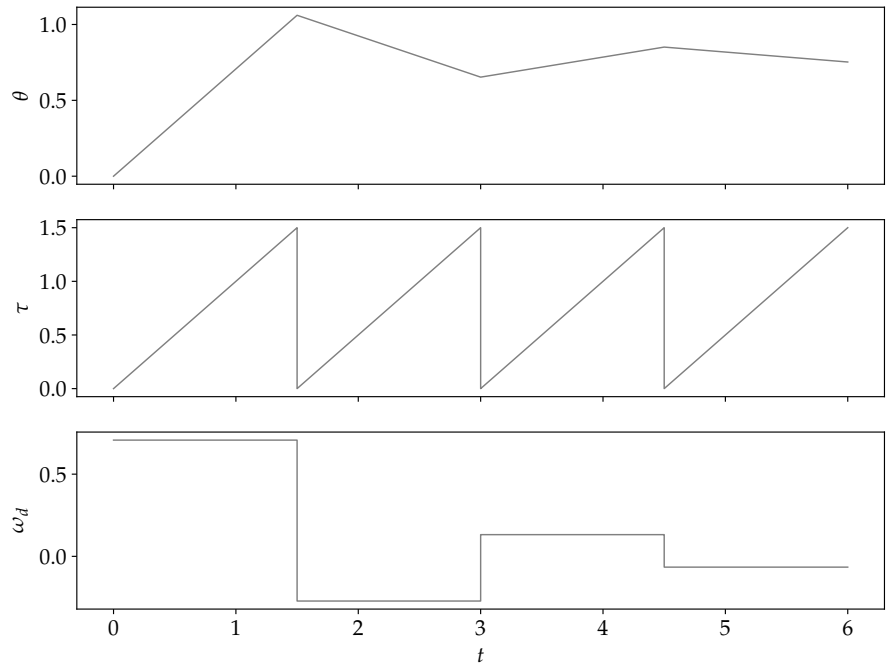


Figure 2.8: Robot's state's components plotted against time

2.3.1 *Background*

Interval analysis is a branch of mathematics where computations are made using a pair of number a and b to represent a real x, such that $a \leq x \leq b$, instead of directly using x. This pair of numbers is called an *interval*. One of the primary motivations for doing so is the ability to enclose errors occurring in floating-point computations and inherent to mathematical models rigorously, and therefore yield guaranteed results. Thus, interval analysis can prove useful when a mathematical proof is wanted for a system or a property of the latter.

Remark 2.7. Since intervals are sets of $\mathbb{R}$, this approach is named set-membership approach. Another widely used method to tackle uncertainties representation is the probabilistic one, which results are given in the form of probability distribution functions (see [121] for a good introduction). While the provided results are then not guaranteed, this approach can be less computationally expensive, and its results are usually easier to interpret.

2.3.1.1 *Round-off errors in computations*

Inside a computer's memory, numbers are usually represented by a finite number of bytes. This means that a computer can use only a finite set of real numbers, all the others "in-between" them being rounded to the nearest representable number. Thus, there exists an error between

When numbers are represented following the IEEE 754 standard, used by many consumer processing units.

the result of a finite precision arithmetic computation and its exact counterpart: the round-off error. The existence of this error can be easily checked in a Python interpreter: If the error remains relatively

```
>>> print(0.1 + 0.2)
  0.30000000000000004
```

Program 2.1: Round-off error in Python

small in this example, they can accumulate and cause a significant difference in the end (see [107] for more examples). Consequently, when a proof based on numerical calculus is wanted, one cannot allow such errors to propagate, whence the use of interval analysis.

Every real number x is then represented in memory by an interval denoted by $[a, b]$ where a and b are both representable numbers inside the computer's memory. Computation is then performed using only representable numbers, yielding an interval which contains the exact result.

2.3.1.2 *Interval analysis in robotics*

Using interval analysis in robotics has another motivation. Using intervals to represent a robot's state vector's components, every measurement made by embedded sensors and every command sent to the actuators will allow performing guaranteed computation and proving statements about a robot's behaviour. The robot's uncertain parameters (friction coefficients, apparent weight...) also ought to be represented as intervals, after a step of parameter estimation [16], [90], [98]. As presented in Section 2.5, interval analysis also allows the introduction of constraints in robotics problems. These are two of the main advantages of using intervals instead of probability distribution functions in robotics. However, the set-membership approach's main drawback is its pessimism: it may prove that a robot lays within a $1\,\mathrm{km}^3$ box, which may not be useful for accurate control.

2.3.2 *Interval arithmetic*

This section briefly introduces the tools and notations used later in this thesis, with a few comprehensive examples. For more details, subtleties of implementation and examples, the reader may refer to [50], [82], [84].

2.3.2.1 *Intervals*

As presented earlier in simpler terms, an *interval* is a connected subset of $\mathbb{R}$. We will denote by $\mathbb{IR}$ the set of all intervals of $\mathbb{R}$. It is represented by a pair of numbers, its upper and lower bounds. Throughout this document, we will denote by $[x]$ an interval of $\mathbb{R}$, and by x^- and x^+ its lower and upper bounds

$$[x] = \left\{ x \in \mathbb{R} \,\middle|\, x^- \leq x \leq x^+ \right\} \tag{2.14}$$

Note that these bounds can be infinite. In most implementations of interval arithmetic, a function yielding the lower/upper bound is provided

$$\mathrm{lb}\left([x]\right) = x^- \tag{2.15}$$
$$\mathrm{ub}\left([x]\right) = x^+ \tag{2.16}$$

Usual set operations can be extended to intervals with care. Let $[x]$ and $[y]$ be two intervals of $\mathbb{R}$.

$$[x] \cap [y] = \{ x \in \mathbb{R} \,|\, x \in [x]\,,\, x \in [y] \} \tag{2.17}$$
$$= \begin{cases} \emptyset & \text{if } x^+ < y^-; \\ \left[\max\left(x^-, y^-\right), \min\left(x^+, y^+\right)\right] & \text{otherwise.} \end{cases} \tag{2.18}$$
$$[x] \sqcup [y] = \left\{ z \in \mathbb{R},\, \min\left(x^-, y^-\right) \leq z \leq \max\left(x^+, y^+\right) \right\} \tag{2.19}$$

Remark 2.8. $\sqcup$ is called the *interval hull* operator, and is the extension of the *union operator* to intervals. It is required, since when $[x] \cap [y] = \emptyset$, $[x] \cup [y]$ is not an interval (it is not connected).

Example 2.9.
- $[-1, 2.1]$, $[0, +\infty]$, $[-\infty, +\infty]$, $[7, 7]$ are intervals;
- $[1, 1] = \{1\} \neq \emptyset$ is a *degenerate* interval;
- $[-2, -1] \cup [1, 2]$ is not an interval, since it is not *connected*;
- $[-2, -1] \sqcup [1, 2] = [-2, 2]$;
- $[-2, 2] \cap [0, 0] = [0, 0]$;
- $[-2, -1] \cap [0, 0] = \emptyset$.

One can define the *midpoint*, the *width* and the absolute value of an interval

$$\mathrm{mid}\left([x]\right) = \frac{x^+ + x^-}{2} \tag{2.20}$$
$$\mathrm{width}\left([x]\right) = x^+ - x^- \tag{2.21}$$
$$|[x]| = \max\left(\left|x^-\right|, \left|x^+\right|\right) \tag{2.22}$$

Usual arithmetic operators can also be defined over $\mathbb{IR}$. Let $\diamond \in \{+, -, \cdot, /\}$, then

$$[x] \diamond [y] = \{x \diamond y \mid x \in [x], y \in [y]\} \tag{2.23}$$

$$\tag{2.24}$$

These operations can be implemented using intervals' bounds. However, special care must be taken, especially with product and division of intervals, because of signs and division by zero that may occur (we refer the reader to [50] for more details).

2.3.2.2 *Interval vectors & matrices*

Intervals can be stacked into vectors, known as *interval vectors* or *boxes*; and concatenated into so-called *interval matrices*. $\mathbb{IR}^n$ denotes the set of axis-aligned boxes of $\mathbb{R}^n$.

The operators and functions defined in Section 2.3.2.1 can be applied component-wise to interval vectors/matrices.

Example 2.10.

$$\text{mid} \begin{bmatrix} [1,2] & [3,4] \\ [-1,1] & [-2,-2] \end{bmatrix} = \begin{bmatrix} 1.5 & 3.5 \\ 0 & -2 \end{bmatrix}$$

$$\text{lb} \begin{pmatrix} [1,2] \\ [-5,0] \end{pmatrix} = \begin{pmatrix} 1 \\ -5 \end{pmatrix}$$

The *norm* of an interval vector $[\mathbf{x}]$ with components $[x_i]$ is defined as follows

$$\| [\mathbf{x}] \| = \max_i \left(|[x_i]| \right) \tag{2.25}$$

Operations between intervals, boxes and interval matrices do not pose any specific difficulty, since they reduce to component-wise and thus scalar interval arithmetic.

The next proposition illustrates the link between the norm of an interval vector and the fact that it belongs to another box.

Lemma 2.1. *Let* $[\mathbf{a}]$ *and* $[\mathbf{b}]$ *be two boxes of* $\mathbb{R}^n$. *Then*

$$[\mathbf{a}] \subset [\mathbf{b}] \implies \| [\mathbf{a}] \| \leq \| [\mathbf{b}] \| \tag{2.26}$$

Proof.

$$\begin{aligned} [\mathbf{a}] \subset [\mathbf{b}] &\iff \forall i \; b_i^- \leq a_i^- \leq a_i^+ \leq b_i^+ \\ &\iff \forall i \; b_i^- \leq a_i^- \leq a_i^+ \leq b_i^+ \leq |[b_i]| \end{aligned}$$

$$
\begin{aligned}
&\iff \forall i \ |[a_i]| \leq |[b_i]| \\
&\iff \forall i \ |[a_i]| \leq |[b_i]| \leq \| [\mathbf{b}] \| \\
&\implies \forall i \ |[a_i]| \leq \| [\mathbf{b}] \| \\
&\implies \max_i |[a_i]| \leq \| [\mathbf{b}] \| \\
&\implies \| [\mathbf{a}] \| \leq \| [\mathbf{b}] \|
\end{aligned}
$$

■

2.3.3 *Interval functions*

Interval functions are the generalisation of real functions to intervals. Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets (e. g. $\mathbb{R}^n$ or $\mathbb{R}^{n\times n}$), and consider the function $f : \mathcal{X} \to \mathcal{Y}$. Then the image of the set $\mathcal{A} \subseteq \mathcal{X}$ is

$$f(\mathcal{A}) = \{f(x) \,|\, x \in \mathcal{A}\}$$

Similarly, the *image set* of an interval $[x]$ (or box or interval matrix) by this function f is

$$f([x]) = \{f(x) \,|\, x \in [x]\} \tag{2.27}$$

The image set might be an interval (or a box, or an interval matrix) with elementary functions, but this is usually not the case, whence the introduction of *inclusion functions*.

Definition 2.7. $[\mathbf{f}]$ is an *inclusion function* for $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ if

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \ \mathbf{f}([\mathbf{x}]) \in [\mathbf{f}]([\mathbf{x}]) \tag{2.28}$$

Therefore, an inclusion function returns an *enclosure* of the image set of $[\mathbf{x}]$ in the form of an interval, interval vector of interval matrix. The interest is that those sets can be much more easily dealt with using the same operators as in interval arithmetic.

Remark 2.9. Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, and a box $[\mathbf{x}] \in \mathbb{IR}^n$. Then not all points in $[\mathbf{f}]([\mathbf{x}])$ have an inverse image inside $[\mathbf{x}]$. This phenomenon is called *wrapping effect*.

For a given function $\mathbf{f}$, one can define multiple different inclusion functions:

- $[\mathbf{f}]$ is *minimal* if for all $[\mathbf{x}] \in \mathbb{IR}^n$, $[\mathbf{f}]([\mathbf{x}])$ is the smallest box containing the set $\mathbf{f}([\mathbf{x}])$. It is then denoted by $[\mathbf{f}([\mathbf{x}])]$. The minimal inclusion function introduces the least wrapping effect in the image box;
- $[\mathbf{f}]$ is *natural* when each variable x of $\mathbf{f}$ is replaced by an interval variable $[x] \ni x$;

- More complex formulations based on the derivatives of $\mathbf{f}$ (see Section 2.3.4).

Usually, finding a minimal inclusion function is not easy, or even impossible (see Section 2.4.4 for more details), whence the use of other formulations... The latter's goal is to find a balance between simplicity (e. g. the natural inclusion is very simple to implement) and precision (i. e. reduced wrapping effect).

Example 2.11. In Figure 2.9 are represented the concepts of image set, inclusion function and minimal inclusion function.

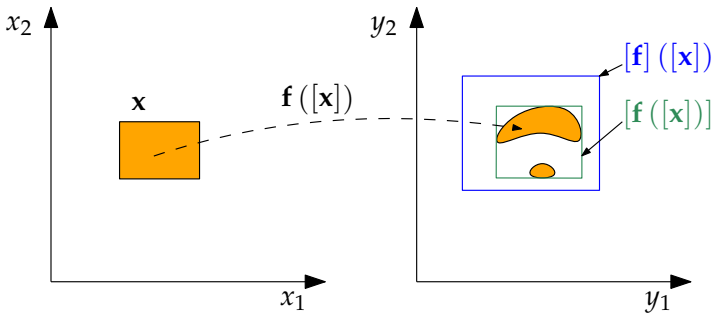


Figure 2.9: Set image, inclusion function and minimal inclusion function

Example 2.12. Let $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = \exp(x) \cdot \sin(x)$. The *natural* inclusion function of f is given by $f([x]) = \exp([x]) \cdot \sin([x])$.

Example 2.13. The goal of this example is to illustrate the wrapping effect occurring when naively composing interval functions. Consider the function given by Equation (2.29), which simply rotates a point of $\mathbb{R}^2$ around the origin by a $\frac{\pi}{4}$ rad angle.

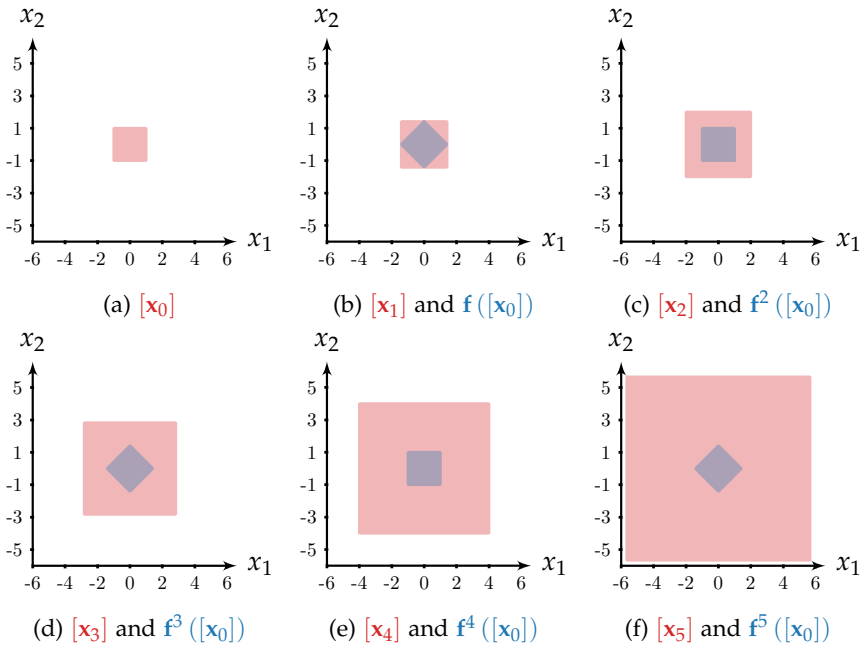
$$\begin{aligned} &\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2 \\ &\mathbf{x} \mapsto \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \cdot \mathbf{x} \end{aligned} \tag{2.29}$$

Let us define the sequences of boxes $[\mathbf{x}_k] = [\mathbf{f}]([\mathbf{x}_{k-1}])$ and $\mathbf{f}^k([\mathbf{x}_0])$ where $[\mathbf{x}_0] = ([-1,1],[-1,1])$. In Figure 2.10, we displayed the first members of both sequences. It is clear that composing an inclusion function can lead to critically overestimate the actual image of the initial box.

This example leads to the following result used later in this thesis.

Lemma 2.2. *Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a function and $[\mathbf{x}_0]$ be a box of $\mathbb{R}^n$. Let us define the sequence $[\mathbf{x}_{k+1}] = [\mathbf{f}]([\mathbf{x}_k])$, where $[\mathbf{f}](\cdot)$ is an inclusion function for $\mathbf{f}$. Then we have*

$$\mathbf{f}^{k+1}([\mathbf{x}_0]) \subseteq \mathbf{f}([\mathbf{x}_k]) \tag{2.30}$$



(a) $[\mathbf{x}_0]$ (b) $[\mathbf{x}_1]$ and $\mathbf{f}([\mathbf{x}_0])$ (c) $[\mathbf{x}_2]$ and $\mathbf{f}^2([\mathbf{x}_0])$

(d) $[\mathbf{x}_3]$ and $\mathbf{f}^3([\mathbf{x}_0])$ (e) $[\mathbf{x}_4]$ and $\mathbf{f}^4([\mathbf{x}_0])$ (f) $[\mathbf{x}_5]$ and $\mathbf{f}^5([\mathbf{x}_0])$

Figure 2.10: Illustration of the wrapping effect due to the composition of inclusion functions

Proof. This lemma is easily derived by induction.

Lemma 2.2 is true for $k = 0$

$$\mathbf{f}^1([\mathbf{x}_0]) = \mathbf{f}([\mathbf{x}_0])$$

Let us assume that it is also true for a given k, we will now prove that it is true for $k+1$.

$$\begin{aligned}\mathbf{f}^{k+1}([\mathbf{x}_0]) \subset \mathbf{f}([\mathbf{x}_k]) &\implies \mathbf{f}^{k+2}([\mathbf{x}_0]) \subset \mathbf{f}(\mathbf{f}([\mathbf{x}_k])) \\ &\implies \mathbf{f}^{k+2}([\mathbf{x}_0]) \subset \mathbf{f}([\mathbf{f}]([\mathbf{x}_k])) \\ &\implies \mathbf{f}^{k+2}([\mathbf{x}_0]) \subset \mathbf{f}([\mathbf{x}_{k+1}])\end{aligned}$$

which concludes this proof. ■

2.3.4 *Centred form*

Let us now introduce the centred form of an interval function, which is an inclusion function for the latter.

Definition 2.8. Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, with a Jacobian matrix $\mathbf{J}_\mathbf{f}$. Then, for all $[\mathbf{x}] \in \mathbb{IR}^n$, the *centred form* $[\mathbf{f}_c]$ associated to $\mathbf{f}$ is given by

$$[\mathbf{f}_c]([\mathbf{x}]) = \mathbf{f}(\mathbf{m}) + [\mathbf{J}_\mathbf{f}]([\mathbf{x}]) \cdot ([\mathbf{x}] - \mathbf{m}) \tag{2.31}$$

where $\mathbf{m} \in [\mathbf{x}]$ (e. g. $\mathbf{m} = \text{mid}([\mathbf{x}])$).

Proposition 2.1. *The centred form is an inclusion function for* $\mathbf{f}$

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) \subset [\mathbf{f}_c]([\mathbf{x}]) \tag{2.32}$$

provided it is defined over $[\mathbf{x}]$.

The main interest of the centred form is that it tends towards the *minimal* inclusion function when the width of $[\mathbf{x}]$ tends towards 0

$$\lim_{\text{width}([\mathbf{x}])\to 0} \frac{\text{width}([\mathbf{f}_c]([\mathbf{x}]))}{\text{width}(\mathbf{f}([\mathbf{x}]))} = 1 \tag{2.33}$$

This means that very little wrapping effect is induced by this formulation when the initial box is small enough.

Example 2.14. Consider the function $f(x) = \cos(x) + \sin(x)$, and the two intervals $[x_a] = [1.2, 1.5]$ and $[x_b] = [1.2, 1.8]$. Denote by m_a and m_b the centres of $[x_a]$ and $[x_b]$ respectively. Using the natural extensions of f, we have

$$[f]([x_a]) = [1.016\ldots, 1.541\ldots]$$
$$[f]([x_b]) = [0.523\ldots, 1.652\ldots]$$

Using the centred form extension of f, we get the enclosures

$$[f_c]([x_a]) = [1.068\ldots, 1.474\ldots]$$
$$[f_c]([x_b]) = [0.427\ldots, 1.779\ldots]$$

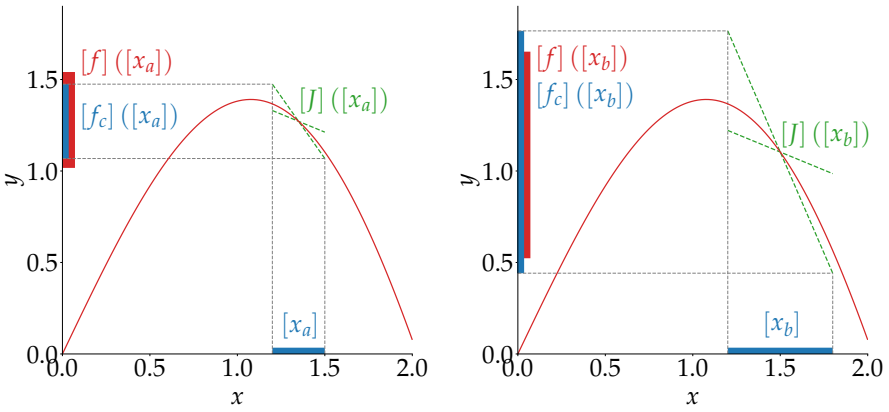
As expected, because $[x_b]$ is not small enough, we have

$$\text{width}([f_c]([x_a])) < \text{width}([f]([x_a]))$$
$$\text{width}([f_c]([x_b])) > \text{width}([f]([x_b]))$$

The results are represented in Figure 2.11. The green dashed lines represent the upper and lower bounds of the Jacobian of f. They are horizontally centred on m_a (or m_b on the second figure), and vertically on $f(m_a)$ (or $f(m_b)$). This allows obtaining $[f_c]([x_a])$ (or $[f_c]([x_b])$) by projection, symbolised by grey dashed lines.

2.4 GUARANTEED INTEGRATION

In Section 2.2, we presented dynamical systems, and in particular continuous-time systems. Those are often represented as a differential equation, which can be integrated over time to determine the system's trajectory from a given initial condition. However, as we pointed out in Section 1.1.4.2, initial conditions and parameters of a robot can be uncertain. Section 2.3 introduced interval analysis, which allows us to represent uncertainty in a guaranteed way. Therefore, in this section, we will present the tools that can be used to integrate an uncertain differential equation in a guaranteed manner, i. e. such that all the solutions are enclosed in the smallest possible box.



(a) Enclosures for $[f]([x_a])$ and $[f_c]([x_a])$

(b) Enclosures for $[f]([x_b])$ and $[f_c]([x_b])$

Figure 2.11: Illustration of the centred form extension

Remark 2.10. As pointed out in Remark 2.7, uncertainties can be represented by probability density functions or random point clouds. While these approaches can be used to obtain a good approximation of the set resulting from the integration, there is no guarantee that it will contain all the possible solutions. Common probabilistic approaches are based on the Monte-Carlo method [78], [121, Chapter 8], which simulates as many trajectories as possible.

We will present the concepts linked to guaranteed integration along the section and illustrate them with a few lines of code using the Computer Assisted Proofs in Dynamics (CAPD) library [120]. We believe that doing so will allow broader use of this library for robotics applications. We will not cover all the functionalities of CAPD, and the reader may refer to [56] for more details.

2.4.1 *Background*

Historically, finding a guaranteed enclosure for the integral of a function is an old problem: Moore and Krückeberg were the first to propose such a method to solve an IVP [60], [81], [82]. The main problem of the algorithms introduced back then was the wrapping effect, which led to the enclosure's explosion after a few integration steps. Algorithms developed since then aimed at reducing this phenomenon to obtain sharper enclosures [28], [46], [75], [87]. More recently, these algorithms have been extended to allow the computation of enclosures for state- and time-derivatives of the system's flow function [134], [135]. Similar algorithms have been adapted to hybrid systems [6], [99], [100].

Usually, these algorithms work with relatively small intervals, to maximise the beneficial effect of the centred form, which they use,

on the wrapping effect's reduction. Therefore, they are used in fields where uncertainties are small and enclose mainly the round-off errors introduced by numerical computations.

2.4.2 *Guaranteed integration algorithms*

Consider the continuous-time dynamical system described by Equation (2.34), with an initial state $\mathbf{x}_0$.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{2.34}$$

In the case of an underwater robot, this initial condition is usually uncertain, and can therefore be represented by a box $[\mathbf{x}_0]$. Additionally, the function $\mathbf{f}$, which describes the system's behaviour over time, uses parameters such as the mass, friction coefficients or even the position of buoyancy elements on the robot, which are usually not entirely known. These parameters can also be represented as intervals. In that case, Equation (2.34) can be rewritten as Equation (2.35) and is called *differential inclusion*.

$$\dot{\mathbf{x}} \in [\mathbf{f}]([\mathbf{x}]) \tag{2.35}$$

Remark 2.11. In the rest of this thesis, the functions used to define the studied systems can be differential inclusions, as long as their uncertain parameters are constant. The reason for that is the following: some algorithms described later need to differentiate the function $\mathbf{f}$ with respect to time, which means that if the uncertain parameters are not constant (and thus their derivatives zero), an enclosure for their derivatives would be required by the differentiation process. Unfortunately, the latter is rarely available in a robotics context.

Example 2.15. Consider a differential drive robot modelled by Equation (2.36), where a_d denotes the desired acceleration, ω_d the desired angular velocity, and $\mathbf{x} = (x, y, \theta, v)$ the robot's state vector.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega_d \\ a_d \end{pmatrix} \tag{2.36}$$

Program 2.2 implements Equation (2.36) using CAPD.

```
1 using capd::autodiff::Node;
2
3 void f(Node t, Node x[], int dimX, Node f[], int dimF, Node
    params[], int dimP) {
4 f[0] = x[3] * cos(x[2]); // ẋ = v · cos(θ)
```

```
5   f[1] = x[3] * sin(x[2]); // ẏ = v · sin(θ)
6   f[2] = params[0]; // θ̇ = ω_d
7   f[3] = params[1]; // v̇ = a_d
8   }
9
10  int main(int argc, char** argv) {
11  int dimX = 4, dimF = 4, dimP = 2;
12  capd::IMap vf(f, dimX, dimF, dimP); // create an interval
      vector field
13  vf.setParameter(0, -0.5); // set the desired angular rate
14  vf.setParameter(1, capd::interval(0.1, 0.2)); // set the
      desired acceleration (here, an interval)
15  capd::IOdeSolver solver(vf, 30); // create a one-step ODE
      solver, using the 30 first derivatives of f
16  capd::ITimeMap timeMap(solver); // create a long-time
      integration solver, built on top of solver
17  // ...
18  return EXIT_SUCCESS;
19  }
20
```

Program 2.2: Implementing a differential equation with CAPD in C++

Note that the developers of CAPD recommend using representable numbers when defining intervals, i. e. write `capd::interval(1, 2) / 10.`. *We will not do that in the rest of this document, for the sake of simplicity.*

A naive solution to integrate this interval differential equation would be to use the natural inclusion function of $\mathbf{f}$, in an Euler integration scheme, as such

$$[\mathbf{x}](t+dt) = [\mathbf{x}](t) + \int_t^{t+dt} [\mathbf{f}]([\mathbf{x}](\tau))\, d\tau \tag{2.37}$$

To implement Equation (2.37) in a computer, and to guarantee the result of the integration, one must then find an enclosure $[\mathbf{x}_g]$ for all the trajectories starting from $[\mathbf{x}](t)$ and evolving for a duration dt. This enclosure is called the *global* enclosure of the solution. Unfortunately, there is no direct way to find this global enclosure, and an iterative approach is often used. Once it is found, Equation (2.37) becomes

$$[\mathbf{x}](t+dt) = [\mathbf{x}](t) + [0, dt] \cdot \mathbf{f}([\mathbf{x}_g]) \tag{2.38}$$

This concept is presented in details in Section 2.4.5.3.

However, another problem arises. Depending on the considered system, one might need better than an Euler integration method to obtain correct results. Most guaranteed integration algorithms can then be split into two groups:

- The ones based on the Taylor expansion method (e. g. [28], [60], [75], [134], [135])
- The ones based on the Hermite-Obreshkov expansion method (see [87])

This thesis will not detail the Hermite-Obreshkov method, but the Lohner algorithm, which is based on the Taylor expansion method

(see Section 2.4.5.2). Since the latter uses successive derivatives of the function $\mathbf{f}$, we give a rough introduction to *automatic differentiation* in Section 2.4.3.

Finally, the representation of the state $[\mathbf{x}]$ is of importance to avoid the wrapping effect. Indeed, an axis-aligned box is not necessarily the best way of enclosing a subset of $\mathbb{R}^n$ (see Section 2.4.4).

2.4.3 *Automatic differentiation*

Consider a function $f : \mathbb{R} \to \mathbb{R}$. The Taylor expansion of the latter around $x \in \mathbb{R}$ is given by Equation (2.39). While formal differentiation of f may be straightforward for the first and second-order, it becomes quite cumbersome as the order grows.

$$f(x+h) = f(x) + \sum_{i=1}^{\infty} \frac{h^i}{i!} \cdot f^{(i)}(x) \tag{2.39}$$

The concept of *automatic differentiation* aims at easing that process. By representing a function and its variables by a tree of simpler components, and using a set of differentiation rules (among which the *chain rule* plays a major role), evaluating a function's successive derivatives becomes a task that a computer can tackle. Automatic differentiation is a field in computer sciences *per se*, so apart from giving a few simple examples, we will not extend much on the matter. We refer the reader to [94], which gives an extensive introduction to the subject.

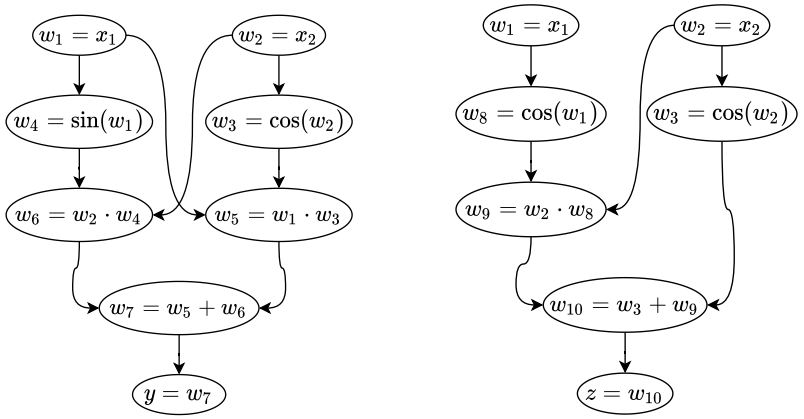
Example 2.16. Consider the function given by Equation (2.40).

$$y = x_1 \cdot \cos(x_2) + x_2 \cdot \sin(x_1) \tag{2.40}$$

The goal is now to represent Equation (2.40) as a binary tree of variables, that we will name after the elements of a sequence w_i.

$$\begin{aligned}
w_1 &= x_1 \\
w_2 &= x_2 \\
w_3 &= \cos(w_2) \\
w_4 &= \sin(w_1) \\
w_5 &= w_1 \cdot w_3 \\
w_6 &= w_2 \cdot w_4 \\
w_7 &= w_5 + w_6 \\
y &= w_7
\end{aligned}$$

This sequence of simple equations can be represented in the computer memory by a binary tree, as depicted in Figure 2.12a. To evaluate y



(a) Representation of Equation (2.40) in a binary tree form

(b) Representation of $\frac{\partial y}{\partial x_1}$ in a binary tree form

Figure 2.12: Binary tree used for automatic differentiation

with specific values of x_1 and x_2, one just needs to descend through the graph until the bottom node.

Now, imagine that we want to evaluate $\frac{\partial y}{\partial x_1}$. The first step is to evaluate each one of the w_i. Then, the solution is obtained by traversing the graph from the bottom up using usual differentiation rules and the chain rule

$$\begin{aligned}
\frac{\partial y}{\partial x_1} &= \frac{\partial y}{\partial w_1} \\
&= \underbrace{\frac{\partial y}{\partial w_7}}_{1} \cdot \frac{\partial w_7}{\partial w_1} \\
&= \underbrace{\frac{\partial w_7}{\partial w_5}}_{1} \cdot \frac{\partial w_5}{\partial w_1} + \underbrace{\frac{\partial w_7}{\partial w_6}}_{1} \cdot \frac{\partial w_6}{\partial w_1} \\
&= \underbrace{\frac{\partial w_1}{\partial w_1}}_{1} \cdot w_3 + \underbrace{\frac{\partial w_3}{\partial w_1}}_{0} \cdot w_1 + \underbrace{\frac{\partial w_2}{\partial w_1}}_{0} \cdot w_4 + \frac{\partial w_4}{\partial w_1} \cdot w_2 \\
&= w_3 + \cos(w_1) \cdot w_2
\end{aligned}$$

This result corresponds to the actual expression of $\frac{\partial y}{\partial x_1}$

$$\frac{\partial y}{\partial x_1} = \cos(x_2) + x_2 \cdot \cos(x_1)$$

Then, since w_1, w_2 and w_3 are known, evaluating $\frac{\partial y}{\partial x_1}$ is straightforward.

This process can be iterated over as many differentiation steps as required. For example, let us propose the set of equations and the binary tree (Figure 2.12b) for $\frac{\partial^2 y}{\partial^2 x_1}$. Pose $z = \frac{\partial y}{\partial x_1}$.

$$w_8 = \cos(w_1)$$

$$
\begin{aligned}
w_9 &= w_8 \cdot w_2 \\
w_{10} &= w_3 + w_9 \\
z &= w_{10}
\end{aligned}
$$

Again, traversing the graph from the bottom up yields

$$
\begin{aligned}
\frac{\partial^2 y}{\partial^2 x_1} &= \frac{\partial z}{\partial w_1} \\
&= \frac{\partial w_{10}}{\partial w_1} \\
&= \underbrace{\frac{\partial w_3}{\partial w_1}}_{0} + \frac{\partial w_9}{\partial w_1} \\
&= \frac{\partial w_8}{\partial w_1} \cdot w_2 + \underbrace{\frac{\partial w_2}{\partial w_1}}_{0} \cdot w_8 \\
&= -\sin(w_1) \cdot w_2
\end{aligned}
$$

which corresponds to the actual expression of $\frac{\partial^2 y}{\partial^2 x_1}$

$$
\frac{\partial^2 y}{\partial^2 x_1} = -\sin(x_1) \cdot x_2
$$

Of course, automatic differentiation can be applied to multidimensional functions and interval-valued functions [84, Chapter 9], [94, Chapters 7-10].

Example 2.17. In the CAPD library, automatic differentiation is available and used according to the user's instructions. Consider the following lines of code, extracted from Program 2.2. One can notice that the

```
1  void f(Node t, Node x[], int dimX, Node f[], int dimF, Node
     params[], int dimP); // define the function f
2  // ...
3  capd::IMap vf(f, dimX, dimF, dimP); // create an interval
     vector field
4  // ...
5  capd::IOdeSolver solver(vf, 30); // implements and ODE solver
     object using the derivative of f to the 30^th order with
     respect to time
6  // ...
7
```

Program 2.3: Automatic differentiation in CAPD

concept of state variable, and more generally of mathematical variable, is enclosed in the `Node` object, for easier manipulation by a computer. In other words, each w_i from the previous examples would be modelled as `Node`, and the functions applied to them reimplemented to yield results in terms of `Node`.

2.4.4 *Interval computation and pessimism*

In this section, we give a brief overview of the existing classes of *representable sets* that can be used to enclose a subset of $\mathbb{R}^n$. The choice of such a class will impact positively or negatively the enclosure yielded by the guaranteed integration algorithm. For a more extensive presentation, the reader may refer to [56], [75], [85].

Remark 2.12. The figures presented below have been obtained using CAPD, by integrating Equation (2.41) with $d = 0.5$ and $[\mathbf{x}_0] = ([0.9, 1.1], [0.4, 0.6])$ during 3 s.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_2 \\ -\sin(x_1) - d \cdot x_2 \end{pmatrix} \tag{2.41}$$

The goal here is not to detail the integration algorithm itself, or even compare the different representable sets in terms of performance, but to display their different components. Let us define the C++ class `MyRepresentableSet`, which defines a representable subset of $\mathbb{R}^n$ and its different operations. Then, the code used to obtain the figures presented below is the following

```
1  void f(Node, Node x[], int, Node f[], int, Node params[], int)
     {
2    f[0] = x[1];
3    f[1] = -sin(x[0]) - 0.5 * x[1];
4  }
5
6  int main(int argc, char** argv) {
7    int dimX = 2, dimF = 2, dimP = 0;
8    capd::IMap vf(f, dimX, dimF, dimP);
9    capd::IOdeSolver solver(vf, 30);
10   capd::ITimeMap timeMap(solver);
11
12   capd::IVector x0 = {{0.9, 1.1}, {0.4, 0.6}}; // Initial state
      of the pendulum
13   MyRepresentableSet set(x0, 0); // Here the definition of the
     chosen representable set, at time t = 0
14   capd::IVector x5 = timeMap(5, set); // box enclosing x(t = 5)
     for all x ∈ [x0]
15   return EXIT_SUCCESS;
16 }
17
```

Program 2.4: Implementation of a damped pendulum system in CAPD

In Figures 2.13 to 2.15 and 2.17, the black shape roughly represents the exact set of solutions of this system. It has been obtained using a Monte-Carlo method and a concave hull algorithm [108]. After

simulating hundreds of the system's trajectories, all initialised in $[\mathbf{x}_0]$, we obtained a point cloud which borders were approximated using this concave hull algorithm.

2.4.4.1 *Interval vectors*

We already introduced this set in Section 2.3.2.2. A box is an axis-aligned subset of $\mathbb{R}^n$, usually given as a vector of intervals. When used in a guaranteed integration algorithm, the box $[\mathbf{x}]$ is usually decomposed into a point vector $\mathbf{c} \in \mathbb{R}^n$ and an interval vector containing $\mathbf{0}$, $[\mathbf{r}] \in \mathbb{IR}^n$. This set is given by

$$[\mathbf{x}] = \mathbf{c} + [\mathbf{r}] \tag{2.42}$$

The main drawback of interval vectors is their inadequacy in representing a set that is not somewhat compact and box-shaped. Indeed, they introduce much wrapping effect in the computations, especially when rotations or non-linear transformations are involved in the computations.

2.4.4.2 *Parallelepipeds*

Parallelepipeds were the first solution developed to solve the wrapping effect problem in computations ([60], [81]). Instead of representing a given set as an axis-aligned box, it is enclosed in a parallelepiped, reducing the wrapping effect significantly (see Figure 2.13). A parallelepiped, denoted by $(\mathbf{c}, \mathbf{B}, [\mathbf{r}])$, is parametrised by a point vector $\mathbf{c}$, a matrix $\mathbf{B}$ and a box $[\mathbf{r}]$ containing $\mathbf{0}$. The corresponding interval vector is defined by

$$[\mathbf{x}] = \mathbf{c} + \mathbf{B} \cdot [\mathbf{r}] \tag{2.43}$$

The box $[\mathbf{x}]$ is displayed in green, the corresponding parallelepiped in blue, and the red point corresponds to $\mathbf{c}$.

Remark 2.13. In the CAPD library, $\mathbf{B}$ is represented as an interval matrix. This simply allows for guaranteed matrix inversion. The elements of $[\mathbf{B}]$ are then extremely thin compared to that of $[\mathbf{r}]$, since they only account for floating-point errors and not the system's uncertainties.

Parallelepipeds allow enclosing sets that do not look like boxes with less wrapping effect, since one degree of freedom is added (the parallelepiped's shape, parametrised by $\mathbf{B}$, which is fixed in the case of a box).

Singularity can be a problem when trying to invert the matrix $\mathbf{B}$, which occurs each time an operation involving two parallelepipeds is executed (we refer the reader to [85] for more details).

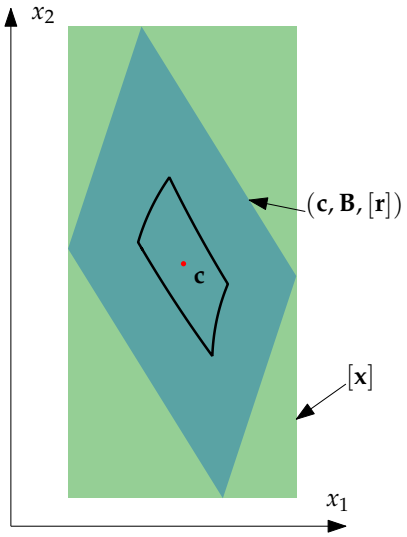


Figure 2.13: Parallelepiped enclosure

The following piece of code is the parallelepiped implementation of the set of solutions of the damped pendulum in CAPD

```
capd::IVector c = {1, 0.5}; // center of [x_0]
capd::IVector r = {{-0.1, 0.1}, {-0.1, 0.1}}; // [x_0] − mid([x_0])
capd::IMatrix B = {{1, 0}, {0, 1}}; // Identity matrix
capd::C0PpedSet set(c, B, r, 0); // Definition of the
    parallelepiped at time t = 0
```

Program 2.5: Implementation of a parallelepiped enclosure in CAPD

2.4.4.3 *Cuboids*

Lohner introduced cuboids in [75] to overcome the potential singularity of the matrix $\mathbf{B}$ in the parallelepiped representation. Lohner advises considering an orthogonal matrix $\mathbf{Q}$. A cuboid is then denoted by $(\mathbf{c}, \mathbf{Q}, [\mathbf{q}])$, and the corresponding interval vector is then given by

$$[\mathbf{x}] = \mathbf{c} + \mathbf{Q} \cdot [\mathbf{q}] \tag{2.44}$$

The box $[\mathbf{x}]$ is drawn in green, the corresponding cuboid in blue, and the red point corresponds to $\mathbf{c}$.

This method loses slightly in terms of wrapping effect compared to the parallelepiped one (see Figure 2.14), but avoids singularity problems in computations.

The following piece of code is the parallelepiped implementation of the set of solutions of the damped pendulum in CAPD

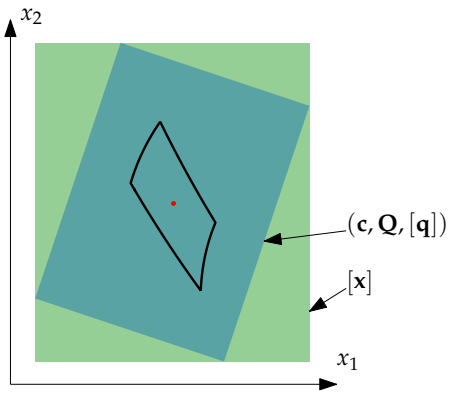


Figure 2.14: Cuboid enclosure

```
capd::IVector c = {1, 0.5}; // center of [x_0]
capd::IVector q = {{-0.1, 0.1}, {-0.1, 0.1}}; // [x_0] - mid([x_0])
capd::IMatrix Q = {{1, 0}, {0, 1}}; // Identity matrix
capd::C0RectSet set(c, Q, q, 0); // Definition of the cuboid at
    time t = 0
```

Program 2.6: Implementation of a cuboid enclosure in CAPD

2.4.4.4 *Doubletons*

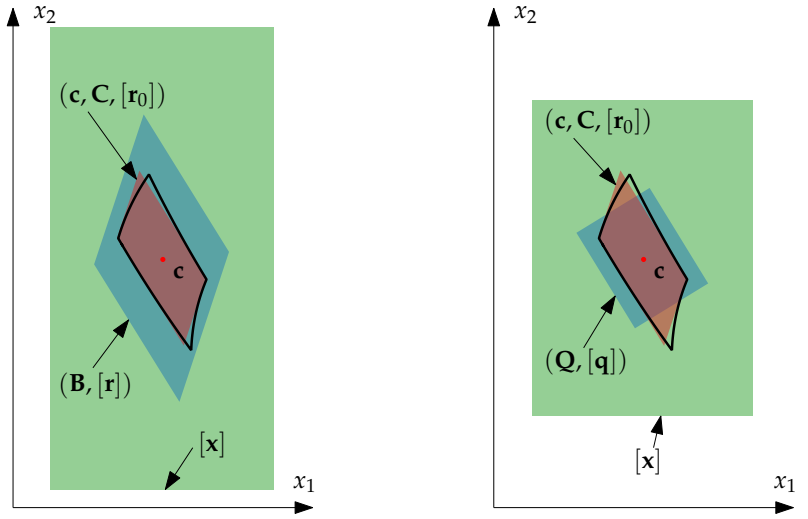
Doubletons have also been introduced in [75] as an extension of cuboids. They allow for a large initial condition (compared to the uncertainties introduced during integration). The doubleton, denoted by $(\mathbf{c}, \mathbf{C}, [\mathbf{r}_0], \mathcal{S})$ is parametrised by a parallelepiped and another representable set $\mathcal{S}$ (being either a parallelepiped or a cuboid)

$$[\mathbf{x}] = \mathbf{c} + \mathbf{C} \cdot [\mathbf{r}_0] + \mathcal{S} \tag{2.45}$$

Figure 2.15a represents a parallelepiped based enclosure, while Figure 2.15b uses a cuboid for $\mathcal{S}$.

For the sake of clarity, Figure 2.16 shows the different steps of computing a doubleton. The shapes drawn with a more intense shade correspond to the actual representable sets, while the less intense, bordered by dashed lines correspond to their respective interval enclosures.

The following piece of code is the cuboid-based doubleton implementation of the set of solutions of the damped pendulum in CAPD



(a) Parallelepiped-based doubleton enclosure

(b) Cuboid-based doubleton enclosure

Figure 2.15: Doubleton enclosures based on parallelepiped and cuboid (identical scale)

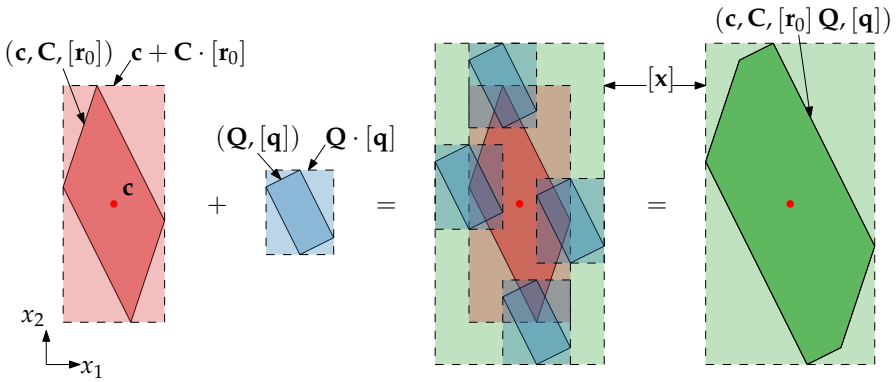


Figure 2.16: Representation of a cuboid-based doubleton

```
capd::IVector c = {1, 0.5}; // center of [x0]
capd::IVector r0 = {{-0.1, 0.1}, {-0.1, 0.1}}; // [x0] − mid([x0])
capd::IMatrix C = {{1, 0}, {0, 1}}; // Identity matrix
capd::IVector q = {0, 0}; // empty q vector
capd::IMatrix Q = {{1, 0}, {0, 1}}; // Identity matrix
capd::C0Rect2Set set(c, C, r0, Q, q, 0); // Definition of the
    cuboid-based doubleton at time t = 0
```

Program 2.7: Implementation of a cuboid-based doubleton enclosure in CAPD

2.4.4.5 *Tripletons*

Tripletons are another way of representing sets, used in CAPD ([56], [120]). A tripleton is a mix between as a cuboid-based and a parallelepiped based doubleton

$$[\mathbf{x}] = \mathbf{c} + \mathbf{C} \cdot [\mathbf{r}_0] + \mathbf{Q} \cdot [\mathbf{q}] \cap \mathbf{B} \cdot [\mathbf{r}] \tag{2.46}$$

where $[\mathbf{Q}]$ is orthogonal. This set is represented in Figure 2.17.

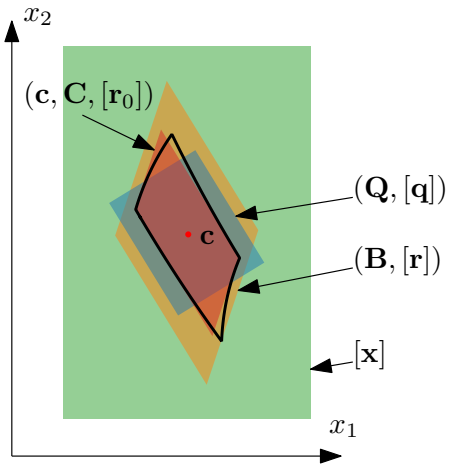


Figure 2.17: Tripleton enclosure

The following piece of code is the tripleton implementation of the set of solutions of the damped pendulum in CAPD. The developer can either define each component of Equation (2.46) manually or directly provide a box for $[\mathbf{x}_0]$ that will be correctly decomposed into the different components.

```
1 capd::IVector x0 = {{0.9, 1.1}, {0.4, 0.6}}; // [x0]
2 capd::C0Rect2Set set(x0, 0); // Definition of the tripleton at
    time t = 0
```

Program 2.8: Implementation of a tripleton enclosure in CAPD

We represented the different steps of computing a tripleton in Figure 2.18.

Remark 2.14. The attentive reader may have noticed that `C0` prefixes each representable set implemented in the CAPD library. This prefix allows computing the solution of an IVP. Other prefixes are available, e. g. `C1` allows solving an IVP and its associated variational equation. More details about these features are available in [56].

In the rest of this thesis, we will only need `C0` and `C1` representable sets.

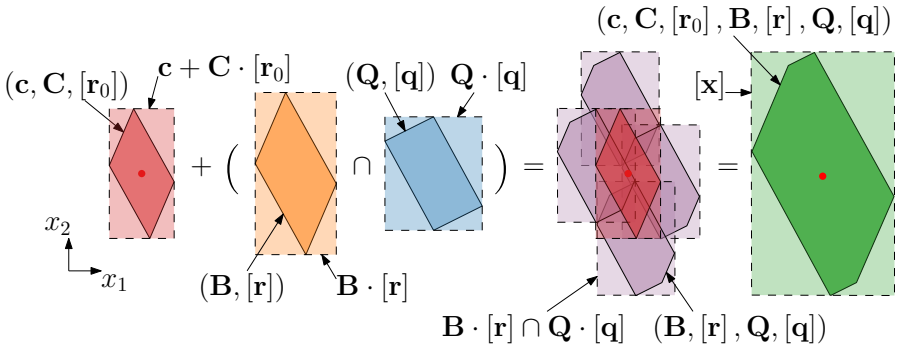


Figure 2.18: Representation of a tripleton

2.4.5 *Lohner's algorithm*

In [73], [75], Lohner presented an algorithm allowing to solve IVPs and boundary value problems rigorously. This subsection aims to give a simple version of this algorithm by explaining and illustrating the different steps to ease its implementation in robotics applications. More details about enclosing the solutions of IVPs can be found in the thesis of Joudrier [55].

We first introduce the algorithm's aim and its operating principle in Section 2.4.5.1; then, we present the latter's theory in Sections 2.4.5.2 to 2.4.5.4. To illustrate the latter, we give a simple implementation (see Section 2.4.5.5) and an example (see Section 2.4.5.6). Finally, we provide some perspectives about Lohner's algorithm in Section 2.4.5.7.

2.4.5.1 *Introduction*

Consider the IVP given by Equation (2.47), where $\mathbf{f} \in \mathcal{C}^p\left(\mathbb{R}^n\right)$, and let us denote by $\phi\left(t, \mathbf{x}\right)$ the flow function of the system.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}\right) \\ \mathbf{x}\left(t_0\right) = \mathbf{x}_0 \end{cases} \tag{2.47}$$

Similarly to a non-rigorous (i. e. not using intervals) method, Lohner's algorithm is capable of integrating Equation (2.47), and thus obtain a sequence of intervals $\left([\mathbf{x}_k]\right)_{k \in \mathbb{N}}$, where k denotes the integration time t_k, such that

$$\forall \mathbf{x}_0 \in [\mathbf{x}_0], \phi\left(t_k, \mathbf{x}_0\right) \in [\mathbf{x}_k]$$

as illustrated in Figure 2.19. Given an interval $[\mathbf{x}] \in \mathbb{IR}^n$ and $t \in \mathbb{R}$, we write

$$\phi_{[\mathbf{x}]}\left(t\right) = \left\{\mathbf{y} = \phi\left(t, \mathbf{x}\right) \mid \mathbf{x} \in [\mathbf{x}]\right\} \tag{2.48}$$

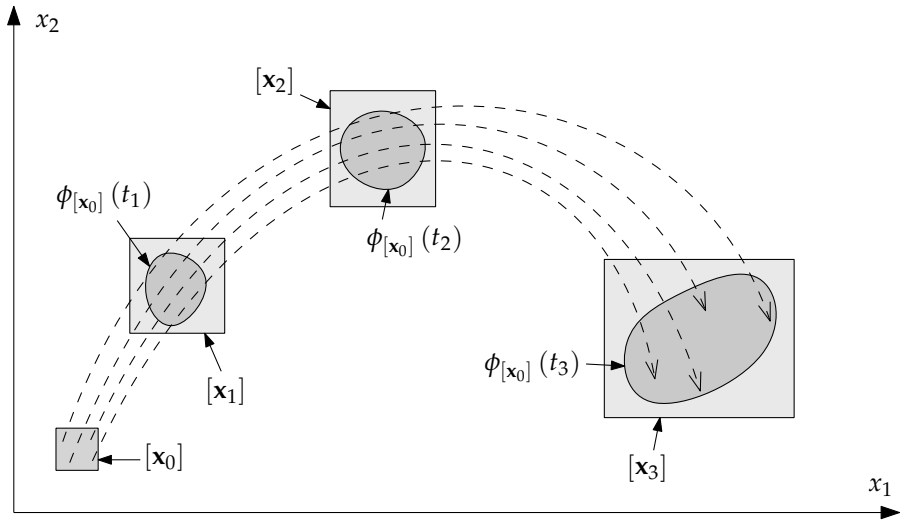


Figure 2.19: Rigorous integration of an IVP

2.4.5.2 *Theory behind Lohner's algorithm*

2.4.5.2.1 TAYLOR-LAGRANGE EXPANSION OF THE SOLUTION AT TIME t_{k+1}

Let $\mathbf{x}(t)$ be a solution of the IVP such that $\mathbf{x}(0) = \mathbf{x}_0 \in [\mathbf{x}_0]$, let $\mathbf{x}_k$ denote $\mathbf{x}(t_k)$, and let h denote the integration step $t_{k+1} - t_k$. Finally, let $[\mathbf{x}_0]$ be the initial enclosure of the solution, such that

$$[\mathbf{x}_0] = \mathbf{x}_0 + [\mathbf{z}_0] \tag{2.49}$$

and denote by $x_{0,j}$, $j \in [1, n]$ the components of $\mathbf{x}_0$.

Since $\mathbf{f} \in \mathcal{C}^p(\mathbb{R}^n)$ and according to Theorem 2.1, we can apply the Taylor-Lagrange's formula to the p^{th} order to the dynamical system $\phi(t, \mathbf{x}_0)$. Doing so around time t_k yields

$$\phi_{\mathbf{x}_0}(t_k + h) = \phi_{\mathbf{x}_0}(t_k) + \sum_{i=1}^{p-1} \frac{h^i}{i!} \phi_{\mathbf{x}_0}^{(i)}(t_k) + \mathbf{z}_{k+1} \tag{2.50}$$

where

$$\phi_{\mathbf{x}_0}^{(i)}(t_k) = \frac{\partial^i \phi_{\mathbf{x}_0}}{\partial t^i}(t_k)$$

and $\mathbf{z}_{k+1}$ denotes the Lagrange remainder. The latter's components are denoted by $z_{k+1,j}$, defined by

$$z_{k+1,j} = \frac{h^p}{p!} \phi_{x_{0,j}}^{(p)}(\tau_{k+1,j})$$

and $\tau_{k+1,j} \in [t_k, t_{k+1}]$. The Lagrange remainder is also called the *discretisation error*: it corresponds to the difference between the true solution at time t_{k+1} and its Taylor series approximation.

To simplify notation, we will let $\mathbf{x}_k^{(i)}$ and $\mathbf{x}^{(i)}$ respectively stand for $\phi_{\mathbf{x}_0}^{(i)}(t_k)$ and $\phi_{\mathbf{x}_0}^{(i)}(t)$. Then, Equation (2.50) can be rewritten as follows

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sum_{i=1}^{p-1} \frac{h^i}{i!}\mathbf{x}_k^{(i)} + \frac{h^p}{p!}\mathbf{x}^{(p)}(\tau_{k+1}) \tag{2.51}$$

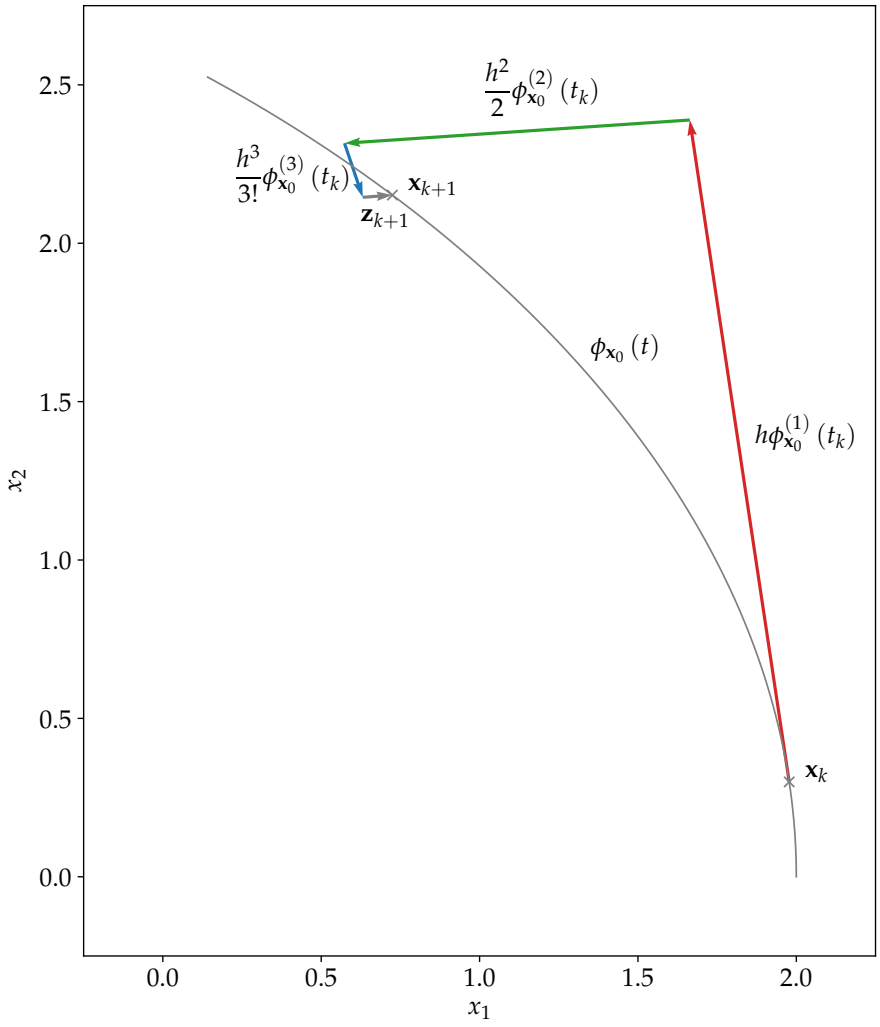


Figure 2.20: Taylor-Lagrange expansion up to order 4, where $\mathbf{z}_{k+1} = \frac{h^4}{4!}\mathbf{x}^{(4)}(\tau_{k+1})$ denotes the Lagrange remainder

As represented in Figure 2.20, Equation (2.51) is an exact expression of $\mathbf{x}_{k+1}$, provided that τ_{k+1} is chosen carefully. This means that using the successive derivatives $\left(\mathbf{x}^{(i)}\right)_{0\leq i\leq p}$, one can find an exact expression of $\mathbf{x}_{k+1}$. The main interest of such an expansion lies in our ability to express those derivatives in terms of $\mathbf{x}$, using automatic differentiation (see Section 2.4.3).

Then, for $q \geq 2$, let us define the function Φ_q:

$$\begin{aligned} \Phi_q : \mathbb{R} \times \mathbb{R}^n &\to \mathbb{R}^n \\ (t, \mathbf{x}) &\mapsto \frac{1}{h} \sum_{i=1}^{q-1} \frac{h^i}{i!} \phi^{(i)}(t, \mathbf{x}) \end{aligned} \tag{2.52}$$

For simplicity of notation, we write $\Phi_q(\mathbf{x}_k)$ instead of $\Phi_q(t_k, \mathbf{x}_0)$. It is worth noticing that Φ_q is $\mathcal{C}^p$ with respect to $\mathbf{x}$, and $\mathcal{C}^{p-q+1}$ with respect to t.

We denote by $\mathbf{z}_{k+1}$ the Lagrange remainder

$$\mathbf{z}_{k+1} = \frac{h^p}{p!} \mathbf{x}^{(p)}(\tau_{k+1}) \tag{2.53}$$

Combining Equations (2.51) and (2.52) yields

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\Phi_p(\mathbf{x}_k) + \mathbf{z}_{k+1} \tag{2.54}$$

2.4.5.2.2 RECURSIVE FORM OF THE TAYLOR-LAGRANGE EXPANSION

Now, one can notice that $\mathbf{x}_{k+1}$ recursively depends on $(\mathbf{z}_i)_{0 \leq i \leq k+1}$, if we choose $\mathbf{z}_0 \in [\mathbf{z}_0]$. Equation (2.54) can then be rewritten:

$$\mathbf{x}_{k+1} = \mathbf{x}_{k+1}(\mathbf{z}_0, \ldots, \mathbf{z}_{k+1}) \tag{2.55}$$

Additionally, $\mathbf{x}_{k+1}$ is continuously differentiable with respect to each one of the $(\mathbf{z}_i)_{0 \leq i \leq k+1}$. To find a tighter enclosure for $\mathbf{x}_{k+1}$, one can then use the mean-value theorem component-wise around the $(\mathbf{m}_i)_{i \in [0, k+1]}$

$$\mathbf{x}_{k+1} = \hat{\mathbf{x}}_{k+1} + \sum_{i=0}^{k+1} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_{k+1}) \cdot (\mathbf{z}_i - \mathbf{m}_i) \tag{2.56}$$

where $\hat{\mathbf{z}}_{k+1}$ denotes the intermediate vectors of the mean-value theorem and $\hat{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1}(\mathbf{m}_0, \ldots, \mathbf{m}_{k+1})$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + h\Phi_p(\hat{\mathbf{x}}_k) + \mathbf{m}_{k+1} \tag{2.57}$$

Remark 2.15. In [75], Lohner proposes to choose the $\mathbf{m}_i$ as the midpoints of the discretisation errors' enclosures, i. e. $[\mathbf{z}_i]$. Since the $\hat{\mathbf{z}}_{k+1}$ are unknown, they may be replaced by their enclosure $[\mathbf{z}_{k+1}]$. Hence the interval counterpart of Equation (2.56) corresponds to the mean-value form of $\mathbf{x}_{k+1}(\mathbf{z}_0, \ldots, \mathbf{z}_{k+1})$

$$[\mathbf{x}_{k+1}] = \hat{\mathbf{x}}_{k+1} + \sum_{i=0}^{k+1} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}([\mathbf{z}_{k+1}]) \cdot ([\mathbf{z}_i] - \mathbf{m}_i) \tag{2.58}$$

Considering that the members of Equation (2.58) are known up to step k, $\hat{\mathbf{x}}_{k+1}$ can be computed using Equation (2.57). Now, let us find a recursive formula for $\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_{k+1})$.

First, rearranging Equation (2.54) in terms of $\hat{\mathbf{z}}_k$ and $\hat{\mathbf{z}}_{k+1}$ yields

$$\mathbf{x}_{k+1}(\hat{\mathbf{z}}_{k+1}) = \mathbf{x}_k(\hat{\mathbf{z}}_k) + h\Phi_p(\mathbf{x}_k) + \mathbf{z}_{k+1} \tag{2.59}$$

Then, consider the case $i \leq k$, and let us differentiate Equation (2.59) with respect to $\mathbf{z}_i$

$$\begin{aligned}
\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_{k+1}) &= \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k) + h\frac{\partial \Phi_p}{\partial \mathbf{z}_i}(\mathbf{x}_k) + \underbrace{\frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{z}_i}}_{\mathbf{0}} \\
&= \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k) + h\frac{\partial \Phi_p}{\partial \mathbf{x}}(\mathbf{x}_k)\frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k) \\
&= \underbrace{\left(\mathbf{I} + h\frac{\partial \Phi_p}{\partial \mathbf{x}}(\mathbf{x}_k)\right)}_{\mathbf{A}_k} \cdot \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k) \\
&= \mathbf{A}_k \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k)
\end{aligned}$$

Now, consider the case $i = k+1$, then we have

$$\begin{aligned}
\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_{k+1}}(\hat{\mathbf{z}}_{k+1}) &= \underbrace{\frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_{k+1}}(\hat{\mathbf{z}}_k)}_{\mathbf{0}} + h\underbrace{\frac{\partial \Phi_p}{\partial \mathbf{z}_{k+1}}(\mathbf{x}_k)}_{\mathbf{0}} + \underbrace{\frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{z}_{k+1}}}_{\mathbf{I}} \\
&= \mathbf{I}
\end{aligned}$$

In conclusion, $\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_{k+1})$ can be computed recursively as follows

$$\frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_{k+1}) = \begin{cases} \mathbf{A}_k \frac{\partial \mathbf{x}_k}{\partial \mathbf{z}_i}(\hat{\mathbf{z}}_k) & \text{if } i \leq k; \\ \mathbf{I} & \text{if } i = k+1. \end{cases} \tag{2.60}$$

where we define

$$\mathbf{A}_k = \left(\mathbf{I} + h\frac{\partial \Phi_p}{\partial \mathbf{x}}(\mathbf{x}_k)\right) \tag{2.61}$$

Now, let us define $\mathbf{A}_{k+1,i} = \dfrac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{z}_i}$. Combining Equations (2.56), (2.57) and (2.60) yields

$$\begin{aligned}
\mathbf{x}_{k+1} &= \hat{\mathbf{x}}_k + h\Phi_p\left(\hat{\mathbf{x}}_k\right) + \mathbf{m}_{k+1} + \sum_{i=0}^{k+1} \mathbf{A}_{k+1,i} \cdot (\mathbf{z}_i - \mathbf{m}_i) \\
&= \hat{\mathbf{x}}_k + h\Phi_p\left(\hat{\mathbf{x}}_k\right) + \mathbf{m}_{k+1} + \sum_{i=0}^{k} \mathbf{A}_{k+1,i} \cdot (\mathbf{z}_i - \mathbf{m}_i) \\
&\qquad + (\mathbf{z}_{k+1} - \mathbf{m}_{k+1}) \\
&= \hat{\mathbf{x}}_k + h\Phi_p\left(\hat{\mathbf{x}}_k\right) + \sum_{i=0}^{k} \mathbf{A}_{k+1,i} \cdot (\mathbf{z}_i - \mathbf{m}_i) + \mathbf{z}_{k+1} \\
&= \hat{\mathbf{x}}_k + h\Phi_p\left(\hat{\mathbf{x}}_k\right) + \mathbf{A}_k \sum_{i=0}^{k} \mathbf{A}_{k,i} \cdot (\mathbf{z}_i - \mathbf{m}_i) + \mathbf{z}_{k+1}
\end{aligned} \tag{2.62}$$

Now, the goal is to find a proper interval evaluation for Equation (2.62). In [75], Lohner gives a sequence of steps which are detailed in the next paragraphs. The first step is to find an enclosure $[\tilde{\mathbf{x}}_k]$ for $\mathbf{x}(t)$ over $[t_k, t_{k+1}]$. This can be seen as a prediction step. The second step (which Lohner splits into substeps for the sake of simplicity) comes down to using the latter to find a local enclosure $[\mathbf{x}_{k+1}]$ such that for every $\mathbf{x}_k \in [\mathbf{x}_k]$, $\phi_{\mathbf{x}_k}(h) \in [\mathbf{x}_{k+1}]$. This is the algorithm's correction step since it contracts the original estimate of $[\tilde{\mathbf{x}}_k]$. This procedure can be iterated over a single time step until reaching a fixed point to contract even more the solution. The principle of the algorithm is illustrated in Figure 2.21.

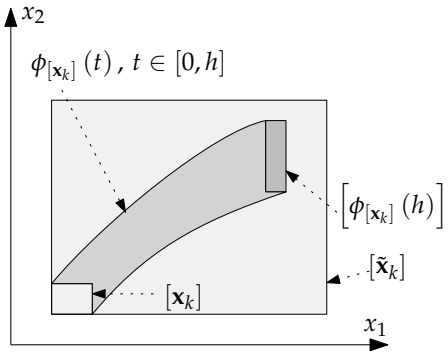


Figure 2.21: Schematic of the Lohner's algorithm

2.4.5.3 *Global enclosure of the solution*

Consider that the algorithm could solve Equation (2.47) up to time t_k. All the quantities required by Equation (2.62) are then known at time t_k or computed using quantities known at time t_k, except for $\mathbf{z}_{k+1}$. Indeed, $\mathbf{z}_{k+1}$ is the discretisation error due to the Taylor-Lagrange expansion and depends on the solution $\mathbf{x}(t)$ at an unknown time value

$\tau_{k+1} \in [t_k, t_{k+1}]$. Since this quantity is unknown, Lohner proposes to find an enclosure $\tilde{\mathbf{x}}_k$ for $\mathbf{x}(t)$ between the times t_k and t_{k+1}

$$[\tilde{\mathbf{x}}_k] = [\{\mathbf{x}(t) \mid t \in [t_k, t_{k+1}], \mathbf{x}(t_k) \in [\mathbf{x}_k]\}] \tag{2.63}$$

This quantity is named *global enclosure* and has been introduced by Moore in [82]. More details about this concept can also be found in [83, Chapter 8], [84, Chapter 10].

This method is called *First-Order Enclosure* (FOE) (as opposed to *high-order enclosure* presented in [23], [89], which is supposed to tackle some of the drawbacks of the FOE method, such as the size of the time step). As explained in [88], the FOE method is based on the Picard-Lindelöf operator and the Banach fixed point theorem.

Definition 2.9. Consider the system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ such that $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$ and $\mathbf{x}(t_k) = \mathbf{x}_k$. The *Picard-Lindelöf operator* of this system is

$$\Pi_{\mathbf{x}}(t) = \mathbf{x}_k + \int_{t_k}^{t} \mathbf{f}(\mathbf{x}(s))\, ds \tag{2.64}$$

Now, consider that $[\mathbf{x}_k]$ is the enclosure of the trajectory $\mathbf{x}(t)$ of the system at time t_k. Then the interval enclosure of the operator is given by

The exponent of $[\tilde{\mathbf{x}}_k^0]$ corresponds to an index, not a power function.

$$\begin{aligned} \Pi_{\mathbf{x}}(t_{k+1}) &= \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(\mathbf{x}(s))\, ds \\ &\in [\mathbf{x}_k] + \int_{t_k}^{t_{k+1}} \mathbf{f}\left([\tilde{\mathbf{x}}_k^0]\right) ds \\ &\subseteq [\mathbf{x}_k] + [0, h]\, \mathbf{f}\left([\tilde{\mathbf{x}}_k^0]\right) \widehat{\in} \left[\tilde{\mathbf{x}}_k^1\right] \end{aligned} \tag{2.65}$$

where $\left[\tilde{\mathbf{x}}_k^0\right]$ is an a priori estimate of $[\tilde{\mathbf{x}}_k]$.

Now, if $\left[\tilde{\mathbf{x}}_k^1\right] \subseteq \left[\tilde{\mathbf{x}}_k^0\right]$, then the Banach fixed-point theorem ensures the existence and uniqueness of the trajectory $\phi_{\mathbf{x}_k}(t)$ such that

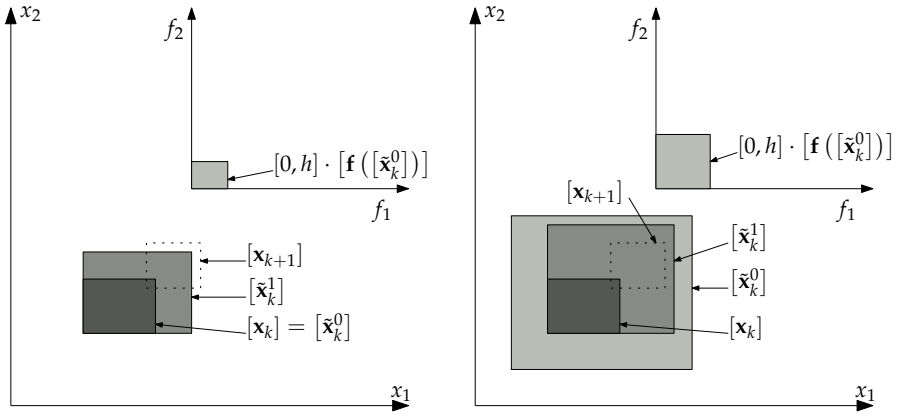
$$\phi_{\mathbf{x}_k}(t) \in \left[\tilde{\mathbf{x}}_k^1\right]$$

for all $t \in [t_k, t_{k+1}]$ and for all $\mathbf{x}_k \in [\mathbf{x}_k]$ (for the derivation, we refer the reader to [88]). Otherwise, one simply needs to inflate $\left[\tilde{\mathbf{x}}_k^0\right]$, say by a factor $\epsilon > 0$, and start over the process.

The problem now lies in the choice of $\left[\tilde{\mathbf{x}}_k^0\right]$. Lohner proposes the strategy presented by Algorithm 1. Depending on the choice of h, this method might not work, hence the automatic reduction of the time step by a factor $\mu \in [0, 1]$ in the algorithm after a given number of iterations N. A graphical interpretation of this algorithm is represented in Figure 2.22.

$[\mathbf{x}_{k+1}]$ is only represented to recall the concept of global enclosure, but is actually unknown by the FOE method.

Therefore, Algorithm 1 returns a global enclosure of the solution $\phi_{\mathbf{x}_k}(t)$ with $t \in [t_k, t_{k+1}]$. Lohner himself indicated that this FOE



(a) First iteration: $[\tilde{\mathbf{x}}_k^1] \not\subset [\tilde{\mathbf{x}}_k^0]$ (b) Second iteration: $[\tilde{\mathbf{x}}_k^1] \subset [\tilde{\mathbf{x}}_k^0]$

Figure 2.22: Global enclosure algorithm

method might reduce the time step much more than a high-order enclosure method, such as the one proposed by [89]. Although it may be a problem due to the computer's limited computational power in certain cases, this algorithm is easy to understand and implement.

Algorithm 1 `GlobalEnclosure (in:` $[\mathbf{x}_k]$`,` h`,` N`,` μ`, 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**
  $n \leftarrow n+1$ *Increase the number of iterations*
  if $n \geq N$ **then**
    $n \leftarrow 0$ *Reset the iteration counter*
    $h \leftarrow \mu \cdot h$ *Reduce the time step by a factor* μ
    Reset the a priori estimates
    $[\tilde{\mathbf{x}}_k^0] \leftarrow [\mathbf{x}_k]$
    $[\tilde{\mathbf{x}}_k^1] \leftarrow [\mathbf{x}_k] + [0,h]\,\mathbf{f}([\tilde{\mathbf{x}}_k^0])$
  end if
  $[\tilde{\mathbf{x}}_k^0] \leftarrow (1+\epsilon)\,[\tilde{\mathbf{x}}_k^1] - \epsilon\,[\tilde{\mathbf{x}}_k^1]$ *Inflate the a priori estimate*
  $[\tilde{\mathbf{x}}_k^1] \leftarrow [\mathbf{x}_k] + [0,h]\,\mathbf{f}([\tilde{\mathbf{x}}_k^0])$ *Compute the new a priori estimate*
end while
return $[\tilde{\mathbf{x}}_k^0]$

2.4.5.4 *Local enclosure of the solution*

Thanks to the global enclosure $[\tilde{\mathbf{x}}_k]$, we can now compute an enclosure for the discretisation error term $\mathbf{z}_{k+1}$, and then focus on the local enclosure of the solution $[\mathbf{x}_{k+1}]$. Here, we will consider that we have access to an enclosure of all the quantities used in Equation (2.62) at time t_k (i. e. $\tilde{\mathbf{u}}_k$, $[\mathbf{x}_k]$, $[\mathbf{z}_k]$, $\mathbf{m}_k$ and $([\mathbf{A}_{k,i}])_{i\in[0,k]}$), and we explain how

to compute their enclosures at time t_{k+1} (i. e. $\tilde{\mathbf{u}}_{k+1}$, $[\mathbf{x}_{k+1}]$, $[\mathbf{z}_{k+1}]$, $\mathbf{m}_{k+1}$ and $[\mathbf{A}_k]$).

2.4.5.4.1 ENCLOSING THE DISCRETISATION ERROR

Now that we have a global enclosure for the trajectories $\phi_{\mathbf{x}_k}(t)$ over $[t_k, t_{k+1}]$, computing an enclosure for the discretisation error $\mathbf{z}_{k+1}$ is straightforward.

Using automatic differentiation, one can express $\mathbf{z}_{k+1}$ in terms of $\mathbf{x}(\tau_{k+1})$, $\tau_{k+1} \in [t_k, t_{k+1}]$. Since $[\tilde{\mathbf{x}}_k]$ is an enclosure for $\mathbf{x}(\tau_{k+1})$, $[\mathbf{z}_{k+1}]$ can in turn be computed.

2.4.5.4.2 ENCLOSING THE TAYLOR SERIES TERMS

The last unknown quantity required by Equation (2.62) to evaluate $[\mathbf{x}_{k+1}]$ is $[\mathbf{A}_k]$ (see Equation (2.61)).

Consider the IVP given by Equation (2.66), where $\mathbf{x}_k \in [\mathbf{x}_k]$, and $[\mathbf{x}_k]$ is the enclosure of the solution of the initial IVP given by Equation (2.47) at time t_k.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \\ \mathbf{x}(t_k) = \mathbf{x}_k \end{cases} \tag{2.66}$$

As presented in Paragraph 2.2.2.3.2, its associated variational equation is given by

$$\begin{cases} \dot{\mathbf{U}} = \dfrac{\partial \mathbf{f}(\mathbf{x}(t))}{\partial \mathbf{x}} \mathbf{U} \\ \mathbf{U}(t_k) = \mathbf{I} \end{cases} \tag{2.67}$$

The p^{th} order Taylor-Young formula applied to the dynamical system ϕ associated to Equation (2.66) around t_k is written as follows

$$\phi(h, \mathbf{x}_k) = \phi(0, \mathbf{x}_k) + \Phi_p(\mathbf{x}_k) + o(h^p) \tag{2.68}$$

Now, differentiating Equation (2.68) with respect to $\mathbf{x}$ yields

$$\begin{aligned} \mathbf{U}(t_k + h) &= \frac{\partial \phi}{\partial \mathbf{x}}(h, \mathbf{x}_k) \\ &= \frac{\partial \phi}{\partial \mathbf{x}}(0, \mathbf{x}_k) + h \frac{\partial \Phi_p}{\partial \mathbf{x}}(\mathbf{x}_k) + o(h^p) \\ &= \underbrace{\mathbf{I} + h \frac{\partial \Phi_p}{\partial \mathbf{x}}(\mathbf{x}_k)}_{\mathbf{A}_k} + o(h^p) \end{aligned} \tag{2.69}$$

Therefore, $\mathbf{A}_k$ corresponds to the p^{th} order Taylor approximation of $\frac{\partial \phi}{\partial \mathbf{x}}(h, \mathbf{x}_k)$ around t_k, and consequently of $\mathbf{U}(t_k + h)$, which yields

$$\mathbf{A}_k = \mathbf{U}(t_k) + h \sum_{i=1}^{p-1} \frac{h^{i-1}}{i!} \mathbf{U}^{(i)}(t_k) \tag{2.70}$$

This expression can be reformulated in terms of $\mathbf{x}(t_k)$ and the successive derivatives of $\mathbf{f}$ with respect to $\mathbf{x}$ at time t_k, which can be computed using automatic differentiation.

Example 2.18. Let us compute the three first time-derivatives of $\mathbf{U}(t_k)$

$$\begin{aligned}
\mathbf{U}(t_k) &= \mathbf{I} \\
\mathbf{U}^{(1)}(t_k) &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k) \cdot \mathbf{U}_{t_k} \\
&= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k) \\
\mathbf{U}^{(2)}(t_k) &= \frac{d}{dt}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k)\right) \cdot \mathbf{U}_{t_k} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k) \cdot \mathbf{U}^{(1)}(t_k) \\
&= \frac{\partial}{\partial \mathbf{x}}\left(\frac{d}{dt}(\mathbf{f}(\mathbf{x}(t_k)))\right) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k)\right)^2 \\
&= \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k) \cdot \mathbf{f}(\mathbf{x}_k)\right) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k)\right)^2 \\
&= \frac{\partial^2 \mathbf{f}}{\partial^2 \mathbf{x}}(\mathbf{x}_k) + 2\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k)\right)^2
\end{aligned}$$

Finally, an enclosure for $\mathbf{A}_k$ can be obtained using Equation (2.70) where each occurrence of $\mathbf{x}_k$ is replaced by its enclosure $[\mathbf{x}_k]$.

2.4.5.4.3 LOCAL ENCLOSURE

The local enclosure $[\mathbf{x}_{k+1}]$ can now be computed using the previous results

$$[\mathbf{x}_{k+1}] = \hat{\mathbf{x}}_k + h\Phi_p(\hat{\mathbf{x}}_k) + [\mathbf{A}_k] \sum_{i=0}^{k} [\mathbf{A}_{k,i}]([\mathbf{z}_i] - \mathbf{m}_i) + [\mathbf{z}_{k+1}] \tag{2.71}$$

However, depending on how the sum in Equation (2.71) is represented, the resulting local enclosure will be more or less tight. This is due to the wrapping effect caused by the product between the sum and the matrix $[\mathbf{A}_k]$. In [73]–[75], Lohner advises representing it as a product of a point matrix $\mathbf{B}_k$ and an interval vector $[\mathbf{r}_k]$ to reduce this wrapping effect (for more details, we refer the reader to Section 2.4.4)

$$\mathbf{B}_{k+1}[\mathbf{r}_{k+1}] = \sum_{i=0}^{k+1} [\mathbf{A}_{k+1,i}]([\mathbf{z}_i] - \mathbf{m}_i) \tag{2.72}$$

where $\mathbf{B}_0 = \mathbf{I}$ and $[\mathbf{r}_0] = [\mathbf{z}_0] - \mathbf{m}_0$. Then, assuming that $\mathbf{B}_{k+1}$ is invertible, this yields

$$\begin{aligned}
[\mathbf{r}_{k+1}] &= \mathbf{B}_{k+1}^{-1} \sum_{i=0}^{k+1} [\mathbf{A}_{k+1,i}] \left([\mathbf{z}_i] - \mathbf{m}_i\right) \\
&= \mathbf{B}_{k+1}^{-1} \sum_{i=0}^{k} [\mathbf{A}_{k+1,i}] \left([\mathbf{z}_i] - \mathbf{m}_i\right) + \mathbf{B}_{k+1}^{-1} \left([\mathbf{z}_{k+1}] - \mathbf{m}_{k+1}\right) \\
&= \mathbf{B}_{k+1}^{-1} [\mathbf{A}_k] \sum_{i=0}^{k} [\mathbf{A}_{k,i}] \left([\mathbf{z}_i] - \mathbf{m}_i\right) + \mathbf{B}_{k+1}^{-1} \left([\mathbf{z}_{k+1}] - \mathbf{m}_{k+1}\right) \\
[\mathbf{r}_{k+1}] &= \left(\mathbf{B}_{k+1}^{-1} [\mathbf{A}_k] \mathbf{B}_k\right) [\mathbf{r}_k] + \mathbf{B}_{k+1}^{-1} \left([\mathbf{z}_{k+1}] - \mathbf{m}_{k+1}\right) \qquad (2.73)
\end{aligned}$$

Due to the matrix inversion appearing in Equation (2.73), which can lead to instability, Lohner advises choosing

$$\mathbf{B}_{k+1} = \mathbf{Q}_{k+1} \qquad (2.74)$$

where $\mathbf{Q}_{k+1}$ corresponds to the orthogonal part of the QR factorisation of the midpoint of $[\mathbf{A}_k] \mathbf{B}_k$

$$\text{mid}\left([\mathbf{A}_k] \mathbf{B}_k\right) = \mathbf{Q}_{k+1} \mathbf{R}_{k+1} \qquad (2.75)$$

Therefore, $\mathbf{B}_{k+1}$ is orthogonal and always invertible.

2.4.5.4.4 CONTRACTING THE GLOBAL ENCLOSURE

Using both $[\mathbf{x}_k]$ and $[\mathbf{x}_{k+1}]$, the global enclosure presented in Section 2.4.5.3 can now be contracted. Doing so and going through the previous steps (enclosing the discretisation error and the Taylor expansion) can yield a tighter enclosure for $[\mathbf{x}_{k+1}]$. This process can be repeated a few times until reaching a fixed point.

The process of contracting the global enclosure is done using the Picard-Lindelöf operator (see Equation (2.65)).

Proposition 2.2. *Let $[\mathbf{x}_k]$ and $[\mathbf{x}_{k+1}]$ denote the local enclosures of the solution $\mathbf{x}(t)$ at times t_k and t_{k+1}. Write $[\tilde{\mathbf{x}}_k]$ the associated global enclosure. Then for all $t \in [t_k, t_{k+1}]$*

$$\begin{cases} \mathbf{x}(t) \in [\mathbf{x}_k] + (t - t_k)\, \mathbf{f}\left([\tilde{\mathbf{x}}_k]\right) \\ \mathbf{x}(t) \in [\mathbf{x}_{k+1}] + (t - t_{k+1})\, \mathbf{f}\left([\tilde{\mathbf{x}}_k]\right) \end{cases} \qquad (2.76)$$

Proof. First, consider $\mathbf{x}_k \in [\mathbf{x}_k]$. According to Definition 2.9, for all $t > t_k$

$$\begin{aligned}
\mathbf{x}(t) &= \mathbf{x}_k + \int_{t_k}^{t} \mathbf{f}(s)\, ds \\
&\in [\mathbf{x}_k] + \int_{t_k}^{t} \mathbf{f}\left([\tilde{\mathbf{x}}_k]\right) ds \\
&\in [\mathbf{x}_k] + (t - t_k)\, \mathbf{f}\left([\tilde{\mathbf{x}}_k]\right)
\end{aligned}$$

Secondly, consider $\mathbf{x}_{k+1} \in [\mathbf{x}_{k+1}]$. Using Definition 2.9 back in time, i. e. for all $t < t_{k+1}$, yields

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}_{k+1} + \int_t^{t_{k+1}} -\mathbf{f}(s)\, ds \\ &\in [\mathbf{x}_{k+1}] - \int_t^{t_{k+1}} \mathbf{f}([\tilde{\mathbf{x}}_k])\, ds \\ &\in [\mathbf{x}_{k+1}] + (t - t_{k+1})\, \mathbf{f}([\tilde{\mathbf{x}}_k])\end{aligned}$$

■

Then, since $[\tilde{\mathbf{x}}_k]$ is an enclosure for $\mathbf{x}(t)$ for $t \in [t_k, t_{k+1}]$, the intersection between the former and both enclosures in Equation (2.76) will provide a tighter global enclosure, which can be used to compute a tighter enclosure $[\mathbf{x}_{k+1}]$, as given by Equation (2.77).

$$[\tilde{\mathbf{x}}_k] \leftarrow [\tilde{\mathbf{x}}_k] \cap [\mathbf{x}_k] + (t - t_k)\, \mathbf{f}([\tilde{\mathbf{x}}_k]) \cap [\mathbf{x}_{k+1}] + (t - t_{k+1})\, \mathbf{f}([\tilde{\mathbf{x}}_k]) \quad (2.77)$$

2.4.5.5 *Implementation of Lohner's algorithm*

Here, we will implement a simple Lohner algorithm, of the form of that presented in [75], using the Taylor-Lagrange expansion formula up to the second-order. This algorithm has been implemented in C++, using the library Ibex (we refer the reader to [17] for more details) and tested on the examples we give below.

We limited our implementation to the second-order because Ibex does not provide automatic differentiation algorithms to the n^{th} order, to the extent of our knowledge. Differentiating further may be useful in some cases. However, doing so in robotics might not always be possible given that the time-derivative of some uncertain parameters of the system might be unknown.

We use the same notations as in Equation (2.47), and we write $[\mathbf{x}_0]$ the initial enclosure of $\mathbf{x}(t = 0)$. To keep the algorithm short and simple, we will assume that the integration step h is constant, and that the global enclosure $[\tilde{\mathbf{x}}_{k+1}]$ is computed at each iteration of the algorithm using Algorithm 1. Also, for simplicity of notation, given a matrix $\mathbf{M}$, we write $\mathrm{Qr}(\mathbf{M})$ the orthogonal part of the QR factorization of $\mathbf{M}$. Finally, we write N the desired number of iterations of the algorithm, and $\mathbf{J}_\mathbf{f}$ the Jacobian of $\mathbf{f}$.

2.4.5.6 *Application*

Here, we will give one example to illustrate the operating principle of Algorithm 2.

Example 2.19.

Example 2.19.1 : *Simplified Lohner algorithm.*
Consider a simple pendulum, such as the one described by Equation (2.9). Let us choose the initial box $[\mathbf{x}_0] = ([0.5, 0.53], [0, 0.03])$. Figure 2.23 has been obtained using Algorithm 2, and represents the first steps of the evolution of the system initialised at $[\mathbf{x}_0]$. Starting from the top right red box, representing $[\mathbf{x}_0]$, the algorithm first estimates

Algorithm 2 `SimpleLohner (in: [x_0], out: [x_N])`

Initialisation:
$\hat{\mathbf{x}}_0 \leftarrow \text{mid}\left([\mathbf{x}_0]\right)$
$[\mathbf{z}_0] \leftarrow [\mathbf{x}_0] - \hat{\mathbf{x}}_0$
$\mathbf{m}_0 \leftarrow \text{mid}\left([\mathbf{z}_0]\right) = \mathbf{0}$
$[\mathbf{r}_0] \leftarrow [\mathbf{z}_0] - \mathbf{m}_0 = [\mathbf{z}_0]$
$\mathbf{B}_0 = \mathbf{I}$
Main loop:
for $k = 0$ **to** $N - 1$ **do**
  $[\tilde{\mathbf{x}}_k], h \leftarrow$ `GlobalEnclosure`$([\mathbf{x}_k], h)$ *see Algorithm 1*
  $[\mathbf{A}_k] \leftarrow \mathbf{I} + h\,[\mathbf{J_f}]\left([\mathbf{x}_k]\right)$
  $[\mathbf{z}_{k+1}] \leftarrow \dfrac{h^2}{2}\,[\mathbf{J_f}]\left([\tilde{\mathbf{x}}_k]\right)[\mathbf{f}]\left([\tilde{\mathbf{x}}_k]\right)$
  $\mathbf{m}_{k+1} \leftarrow \text{mid}\left([\mathbf{z}_{k+1}]\right)$
  $\mathbf{B}_{k+1} \leftarrow \text{Qr}\left(\text{mid}\left([\mathbf{A}_k]\,\mathbf{B}_k\right)\right)$
  $[\mathbf{r}_{k+1}] \leftarrow \left(\mathbf{B}_{k+1}^{-1}\,[\mathbf{A}_k]\,\mathbf{B}_k\right)[\mathbf{r}_k] + \mathbf{B}_{k+1}^{-1}\left([\mathbf{z}_{k+1}] - \mathbf{m}_{k+1}\right)$
  $\hat{\mathbf{x}}_{k+1} \leftarrow \hat{\mathbf{x}}_k + h\mathbf{f}\left(\hat{\mathbf{x}}_k\right) + \mathbf{m}_{k+1}$
  $[\mathbf{x}_{k+1}] \leftarrow \hat{\mathbf{x}}_{k+1} + \mathbf{B}_{k+1}\,[\mathbf{r}_{k+1}]$
end for
return $[\mathbf{x}_N]$

the top right green a priori enclosure, thanks to which it computes the orange cuboid and finally the blue box representing $[\mathbf{x}_1]$. For the sake of clarity, we jumped a few integration steps before displaying $[\mathbf{x}_5]$, $[\mathbf{x}_6]$ and their a priori enclosure. The red dotted line is the trajectory of $\text{mid}\left([\mathbf{x}_0]\right)$.

Example 2.19.2 : *CAPD integration.*
This example aims to compare qualitatively the results obtained by our simplified version of the Lohner algorithm, and its CAPD equivalent (i. e. using a cuboid enclosure and a second-order Taylor expansion).

So far, the pieces of code we provided only showed how to perform long-time integration, without the possibility of access to the solution's enclosures before the final time of the integration. Program 2.9 allows doing so, and in particular to obtain Figure 2.24. The differences between Figures 2.23 and 2.24 are due to a series of improvements and optimisations made by the developers of CAPD, e. g. the automatic time-step adjustment.

2.4.5.7 *Conclusion*

In this section, we introduced, explained and illustrated the algorithm given by Lohner in [75]. This algorithm is part of the long development of guaranteed integration methods, which have led to the development of libraries such as CAPD. While more performing algorithms exist

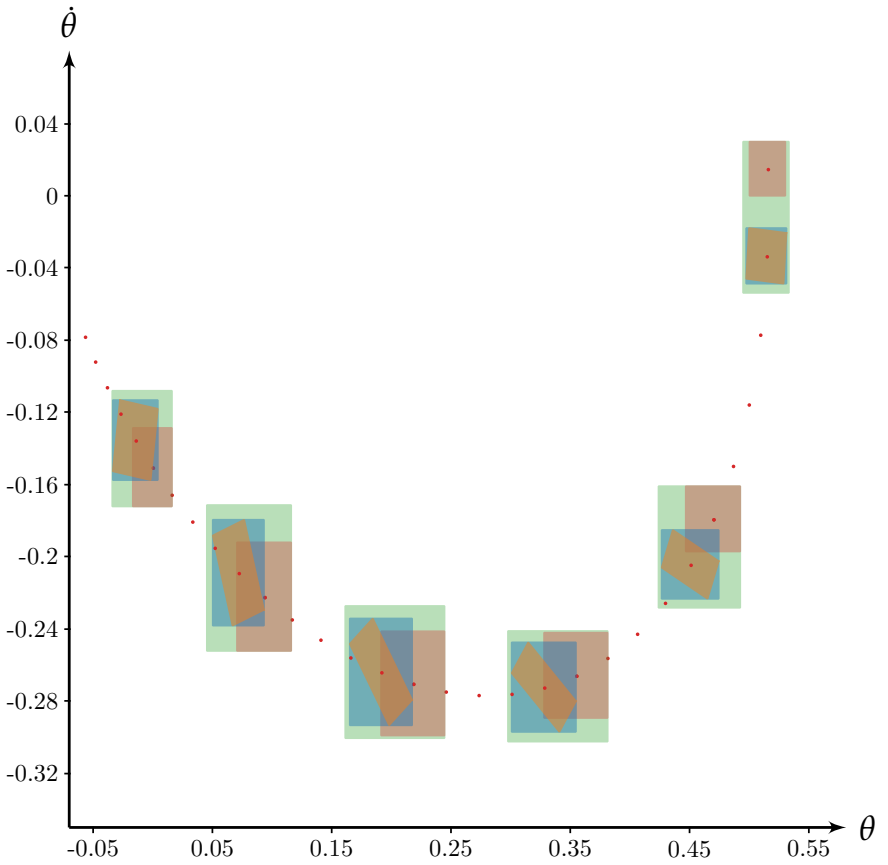


Figure 2.23: Integration of a simple pendulum by Algorithm 2

```
1    double finalTime = 3.0, dt = 0.1;
2    // ...
3    ITimeMap timeMap(solver);
4    capd::IVector x0 = {{0.5, 0.53}, {0, 0.03}};
5    capd::C0RectSet set(x0, 0);
6    ITimeMap::SolutionCurve curve(initTime); // allows accessing
     the enclosure at any time between 0 and finalTime
7    timeMap(finalTime, x0, curve);
8
9    for (int i = 1; i < 30; ++i) {
10     capd::IVector x = curve(i * dt); // returns the enclosure
     of x(t = i · dt)
11   }
12
```

Program 2.9: Guaranteed integration of a damped pendulum with CAPD

nowadays, it seemed important to introduce this one in a robotics context, where it is not widely known. We believe that it can lead to improvements in various existing interval analysis libraries used in a robotics context ([17, Ibex], [103, Tubex]) since it allows for sharper guaranteed integration than the existing algorithms. In particular,

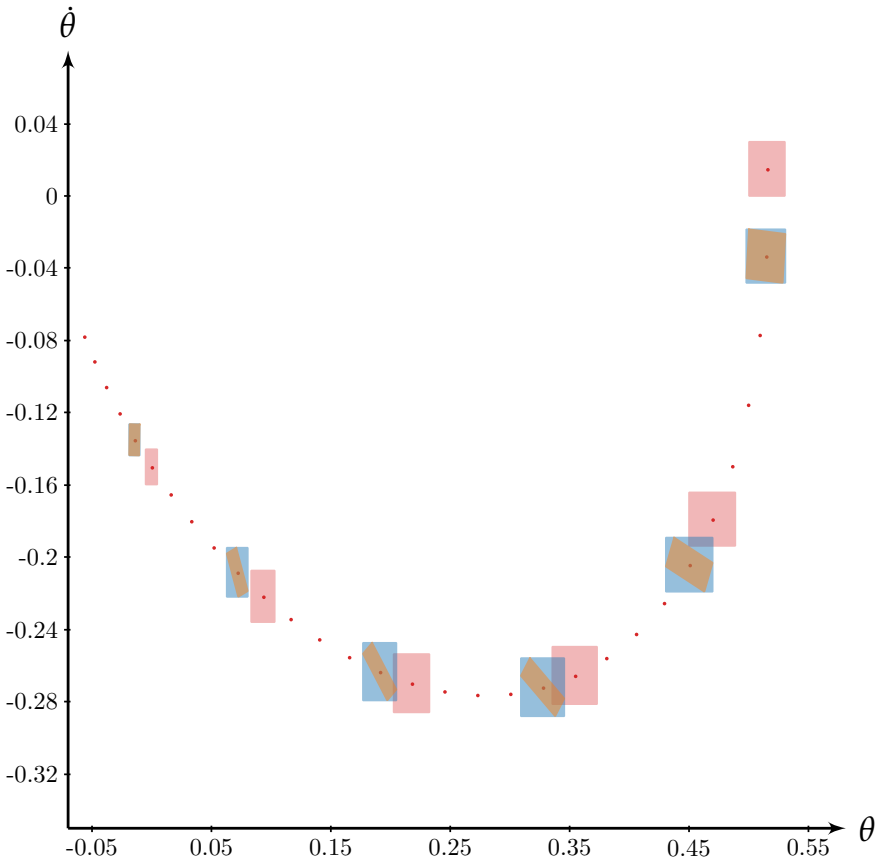


Figure 2.24: Integration of a simple pendulum by Program 2.9 (the global enclosure is not displayed because it is not easily accessible in CAPD.)

Section 4.3 adapts this algorithm to the Tubex library and transforms it into a contractor, to standardise its use.

However, there are constraints that one should keep in mind while using such an algorithm: it only performs well because of the small size of the intervals involved in the computations (since it uses the centred form to reduce the wrapping effect). Handling small intervals in a robotics context might not always be possible (sometimes, the uncertainties are too large), and using paving algorithms might be necessary (see Section 2.6).

2.5 CONSTRAINT SATISFACTION PROBLEMS AND CONTRACTORS

A successful approach to deal with underwater robot localisation is based on contractors [50, Chapter 4]. Its principle is the following: assuming a priori that the robot is located inside a very large box (say, the ocean's size), the enclosure for its actual position is then improved using constraints thanks to which impossible positions are

removed. This approach allows dealing with far larger intervals than usual guaranteed enclosure algorithms. Chapter 4 will explore the possibilities of combining both approaches to solve problems in a guaranteed way.

In this section, we will introduce the concepts of *Constraint Satisfaction Problem (CSP)* and *contractors*. We will not introduce each existing contractor, but merely give a simple example to illustrate the definitions of the concepts mentioned above. More details and examples about CSP and contractors can be found in [18, Chapter 3], [50, Chapter 4].

Definition 2.10. Consider a triplet $\mathcal{H} = (\mathbf{x}, \mathcal{L}, [\mathbf{x}])$ such that

1. $\mathbf{x} \in \mathbb{R}^n$ is a set of variables;
2. $\mathcal{L}$ is a set of constraints $\{\mathcal{L}_{f_1}, \ldots, \mathcal{L}_{f_m}\}$, where $f_i : \mathbb{R}^n \to \mathbb{R}$ is the function describing the constraint $f_i(\mathbf{x}) = 0$;
3. $[\mathbf{x}] \in \mathbb{IR}^n$ is the set of domains for $\mathbf{x}$.

The *solution set* of $\mathcal{H}$ is defined as

$$\mathcal{S} = \{\mathbf{x} \in [\mathbf{x}] \,|\, \forall i,\, f_i(\mathbf{x}) = 0\}$$

Solving the *Constraint Satisfaction Problem* associated to $\mathcal{H}$ consists in finding its solution set.

Example 2.20. Imagine an USV equipped with a compass and a camera and measuring the bearings of three lighthouses α_1, α_2 and α_3 (see Figure 2.25).

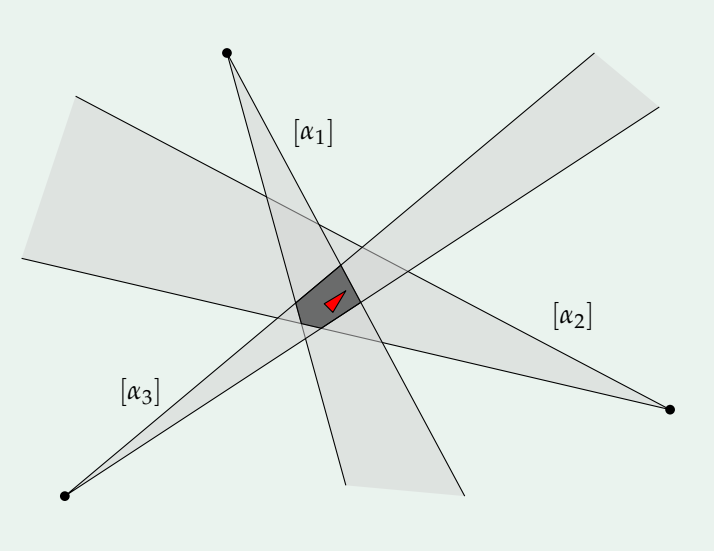


Figure 2.25: Triangulating a robot

These measurements are not perfect and are thus enclosed in three intervals $[\alpha_1]$, $[\alpha_2]$ and $[\alpha_3]$. Using the principle of triangulation, the

USV can determine its position (given that the lighthouses' positions are known, and that the latter are distinguishable from one another).

This localisation problem can be formalized using a CSP

1. Let x and y be the position of the USV in a cartesian plane, then set $\mathbf{x} = (x, y, \alpha_1, \alpha_2, \alpha_3)$.
2. Let (x_1, y_1), (x_2, y_2) and (x_3, y_3) be the positions of the lighthouses. Then let us define the three following constraints

$$\mathcal{L}_{f_1} : f_1(\mathbf{x}) = \tan\alpha_1 - \frac{y_1 - y}{x_1 - x} = 0$$
$$\mathcal{L}_{f_2} : f_2(\mathbf{x}) = \tan\alpha_2 - \frac{y_2 - y}{x_2 - x} = 0$$
$$\mathcal{L}_{f_3} : f_3(\mathbf{x}) = \tan\alpha_3 - \frac{y_3 - y}{x_3 - x} = 0$$

3. The set of domains for $\mathbf{x}$ is as follows

$$[\mathbf{x}] = ([-\infty, \infty], [-\infty, \infty], [\alpha_1], [\alpha_2], [\alpha_3])$$

In Figure 2.25, the green box represents the domain for x and y, while each angular domain corresponds to a bearing measurement. The resulting set of solutions, in dark grey, is obtained via *contraction* of the CSP (see below).

Now, finding the solution of a CSP might not always be possible, according to [50, Chapter 4]. Instead, it is possible to find an enclosure for $\mathcal{S}$ by contracting the CSP $\mathcal{H}$, i. e. finding a domain $[\mathbf{y}]$ such that $\mathcal{S} \subset [\mathbf{y}] \subset [\mathbf{x}]$. To do so, each constraint can be transformed into a contractor, which is an operator used to contract the set $[\mathbf{x}]$, i. e. remove the solutions from $[\mathbf{x}]$ that do not satisfy that constraint.

Definition 2.11. A *contractor* associated to a constraint $\mathcal{L}_f$ of a CSP $\mathcal{H}$ with a solution set $\mathcal{S}$ is an operator $\mathcal{C}_f : \mathbb{IR}^n \to \mathbb{IR}^n$ such that

1. $\forall [\mathbf{x}] \in \mathbb{IR}^n$, $\mathcal{C}_f([\mathbf{x}]) \subset [\mathbf{x}]$
2. $\forall [\mathbf{x}] \in \mathbb{IR}^n$, $[\mathbf{x}] \cap \mathcal{S} \subseteq \mathcal{C}_f([\mathbf{x}])$

One of the main interests of contractors is that they can be combined through operations (e. g. intersection, union, composition...) to yield a smaller enclosure for $\mathcal{S}$.

Example 2.21. Consider the previous example. Each constraint splits the $\mathbb{R}^2$ plane into two zones: the one where it is satisfied and the rest. Assuming that the lighthouses have an infinite range, and if we only consider axis-aligned boxes of $\mathbb{R}^2$, we have drawn in Figure 2.26 the result of the initial domain contraction $[\mathbf{x}]$.

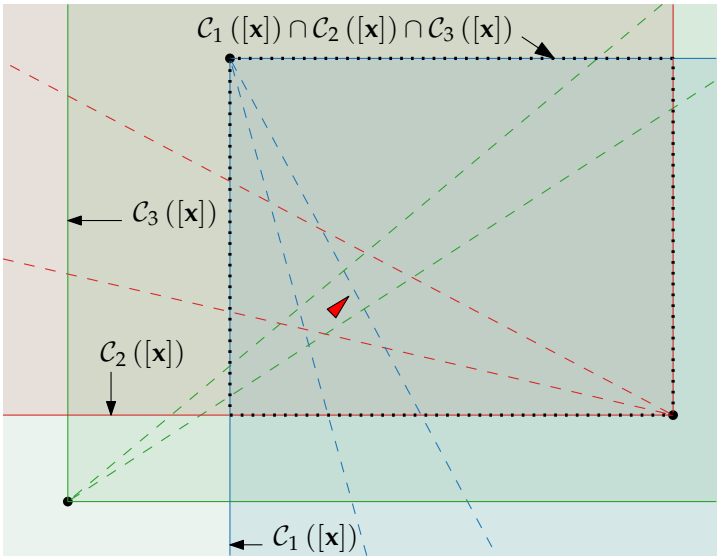


Figure 2.26: Using contractors to localise a robot

In this thesis, we will present two new contractors. The first one (see Section 3.4.1) is adapted to stability analysis and does not aim to be used as a contractor for CSP. The second one (see Section 4.3) adapts the Lohner algorithm to contract tubes, which are intervals of trajectories, i. e. subsets of $\mathbb{R} \times \mathbb{R}^n$. It corresponds to a so-called differential constraint, i. e. the one introduced by a differential equation.

2.6 PAVING ALGORITHMS

Sometimes, especially in robotics, the uncertainties might be too large to handle for integration algorithms or contractors, i. e. they introduce too much wrapping effect in the computations, no matter what intelligent algorithm is used to reduce the latter. A handy method to get a better approximation for a set is to use paving algorithms [50, Chapter 3] as a last resort. Their working principle is straightforward: instead of using one large box to enclose a set, the latter is *paved* by a list of non-overlapping smaller boxes, which allows enclosing much more complex sets. Algorithms using this paradigm have been implemented to perform accurate set inversion [53] or used jointly with contractors [18] to get a more accurate approximation of the result set (see Figure 2.27). Usually, two approximations can be given when characterizing a set $\mathcal{S}$: the inner one (in red in Figure 2.27) and the outer one (both the red and yellow boxes in Figure 2.27).

Despite being very useful, these algorithms are only used as a last resort and in low dimensional systems. Indeed, each new dimension is a new one to pave, increasing the number of boxes to be treated and the need for computational power and processing time. There are ways

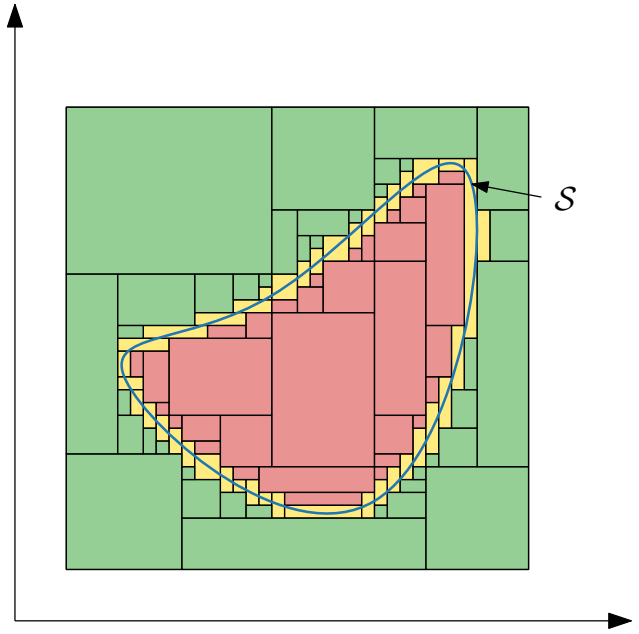


Figure 2.27: Inner and outer paving of a set $\mathcal{S}$: red boxes are inside $\mathcal{S}$, green ones outside, and nothing can be said about yellow boxes

to pave "intelligently" a set, to avoid useless additional computations e. g. using a bisection strategy coupled with a binary tree to store resulting boxes. This thesis's contributions are not based on paving algorithms, the latter are only used to illustrate the methods presented later.

3

PROVING FEASIBILITY OF A DOCKING MISSION : APPROACH BY STABILITY ANALYSIS

TABLE OF CONTENTS

3.1 INTRODUCTION

This chapter is devoted to the first approach we explored to solve the docking problem. The outline of the problem is simple: we want to prove that a specific robot can dock onto a specific target.

On the one hand, the robot embeds various sensors and actuators, managed by a computer and computer programs. All of these are known to us in details: we have a model and a set of identified parameters for each component, possibly under the form of intervals. On the other hand, the target can be a garage equipped with a docking station deployed underwater, or hanging from a cable attached to a surface vessel; or even another robot such as an AUV cruising in the ocean. What is important is for us to have access to its model and its state, which implies that the target is cooperating with the robot.

Then, by combining the different models of the couple robot/target, we can get a global model describing the latter, parametrised by values known more or less precisely. The fact that the robot reaches the target translates to the convergence of one or more state variables towards a given value. For example, the position of the robot's docking interface and the target's one should end up being the same. Mathematically speaking, we have just described a stable system: starting from an initial point in its state space, the dynamical system formed by the couple robot/target should converge towards a stable point.

This leads us to the introduction of Section 3.2, in which we recall the concepts linked to stability of dynamical systems, starting from the definition of the latter to the existing methods for proving stability of a system, whether continuous, discrete or hybrid. Then, in Section 3.3, we formalise the docking problem as a stability problem, using the concepts introduced earlier. Finally, in Sections 3.4 and 3.5, we introduce the tools we created while developing our approach, we present methods that allow us to prove stability of discrete-time, continuous-time and hybrid uncertain systems and we illustrate the latter through comprehensive examples.

3.2 STABILITY OF DYNAMICAL SYSTEMS

This section presents some preliminaries about stability of dynamical systems, and the methods traditionally used to prove stability for a dynamical system, whether continuous- or discrete-time. We also briefly present the case of hybrid systems.

3.2.1 *Stability of continuous-time dynamical systems*

3.2.1.1 *Stability according to Lyapunov*

We graphically introduced the concept of stability in Figure 1.9. In this example, the marble would oscillate around the bottom of the valley before immobilising there. If the marble were slightly shifted to either side, it would come back to that point. Now, it is straightforward that the marble's behaviour in a neighbourhood of a given equilibrium point depends on the shape of the cloth, the position and velocity of the marble, and many other physical parameters. The lowest point of the valley is a *stable equilibrium state*.

Consider the autonomous system described by Equation (3.1), where $\mathbf{f} \in \mathcal{C}^k(\mathbb{R}^n)$.

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

A state $\bar{\mathbf{x}}$ is an *equilibrium point* if $\mathbf{f}(\bar{\mathbf{x}}) = \mathbf{0}$, i. e. if the system remains still when initialised at $\bar{\mathbf{x}}$. To study equilibrium points, and assess their stability, one usually checks the linearised system's eigenvalues at the point $\bar{\mathbf{x}}$, given by Equation (3.2).

$$\dot{\mathbf{x}} = \mathbf{J}_{\mathbf{f}}(\bar{\mathbf{x}}) \cdot \mathbf{x} \tag{3.2}$$

Figure 3.1 shows a few possible behaviours for a planar system, depending on its eigenvalues [40, Chapter 13], [41, Chapter 8]. The

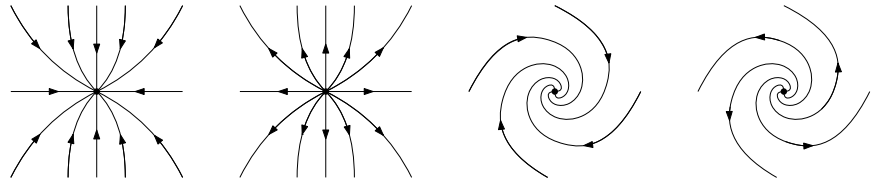


Figure 3.1: Different types of equilibria, where z corresponds to the eigenvalues of the system

equilibria represented in Figure 3.1 are called *hyperbolic* since the real parts of their eigenvalues are all non-zero.

Let us now give the usual definitions of stability of a continuous-time system [59, Chapter 4] (initially proposed by Lyapunov in [72]).

Definition 3.1 (Stability)**.** The state $\bar{\mathbf{x}}$ is *stable* for the system described by Equation (3.1) if

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \|\mathbf{x}(0) - \bar{\mathbf{x}}\| < \delta \implies \forall t > 0,\ \|\mathbf{x}(t) - \bar{\mathbf{x}}\| < \varepsilon \tag{3.3}$$

Otherwise, it is *unstable*.

This definition implies that if the system is initialised close enough to the equilibrium point $\bar{\mathbf{x}}$, it will remain in its neighbourhood.

Definition 3.2 (Asymptotic stability)**.** The state $\bar{\mathbf{x}}$ is *asymptotically stable* for Equation (3.1) if it is *stable* and the following property holds.

$$\exists \delta > 0,\ \|\mathbf{x}(0) - \bar{\mathbf{x}}\| < \delta \implies \lim_{t \to \infty} \mathbf{x}(t) = \bar{\mathbf{x}} \tag{3.4}$$

This definition is stronger than the previous one since it implies that the system will reach the state $\bar{\mathbf{x}}$ after an infinite time.

Definition 3.3 (Exponential stability)**.** The state $\bar{\mathbf{x}}$ is *exponentially stable* for Equation (3.1) if there exist $\alpha > 0$ and $\beta > 0$ such that

$$\exists \delta > 0,\ \|\mathbf{x}(0) - \bar{\mathbf{x}}\| < \delta \implies \|\mathbf{x}(t) - \bar{\mathbf{x}}\| \leq \alpha \|\mathbf{x}(0) - \bar{\mathbf{x}}\| e^{-\beta \cdot t} \tag{3.5}$$

This definition implies that the system converges faster than a decreasing exponential. It also implies asymptotic stability.

The definitions given above are illustrated in Figure 3.2 for a one-dimensional system, to grasp the meaning of the neighbourhoods ε and δ.

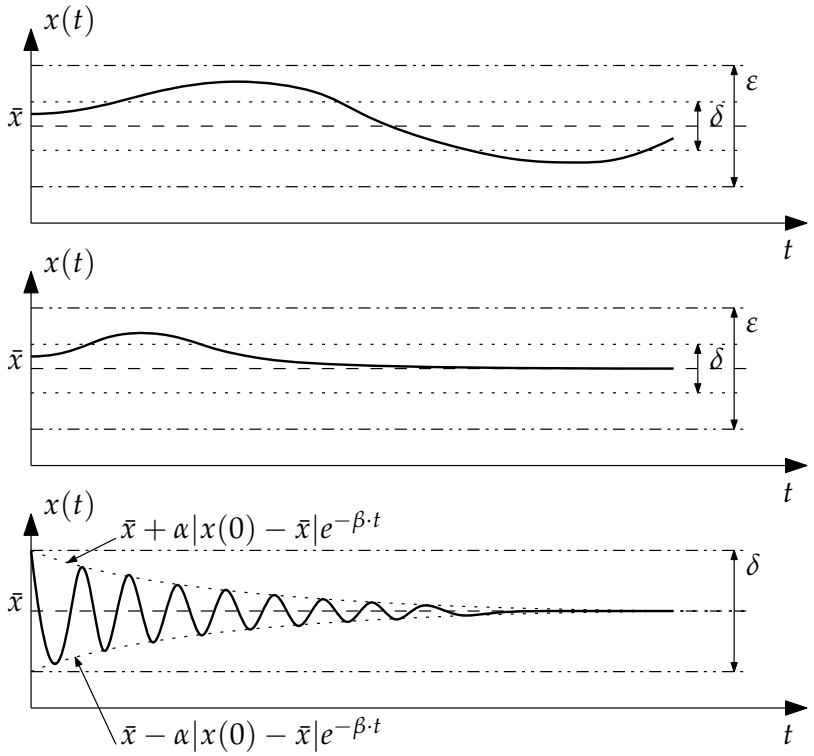


Figure 3.2: Illustration of Definition 3.1 (top), Definition 3.2 (middle), and Definition 3.3 (bottom) on a one-dimensional system: its trajectory remains in a given neighbourhood around the equilibrium point $\bar{x}$

Example 3.1. Hyperbolic equilibrium points having only eigenvalues with negative real parts are asymptotically stable, contrary to hyperbolic equilibrium points that are unstable.

3.2.1.2 *Proving stability using Lyapunov's theorem*

What happens when of the linearised system's eigenvalues are pure imaginary, i. e. the equilibrium point is not hyperbolic? In that case, the method widely used to determine stability is the one developed by Lyapunov [41, Chapter 9], [59, Chapter 4]. This method is based on Theorem 3.1, which is illustrated in Example 3.2.

Theorem 3.1. *Consider the system described by Equation* (3.1)*, and an equilibrium point* $\bar{\mathbf{x}}$ *of the latter. If there exists a function* $V : \mathbb{R}^n \to \mathbb{R}$ *such that* $V(\bar{\mathbf{x}}) = 0$ *and* $V(\mathbf{x}) > 0$ *in a neighbourhood* E *of* $\bar{\mathbf{x}}$*, then*

- *if* $\frac{dV}{dt}(\mathbf{x}) \leq 0$ *for all* $\mathbf{x} \in E$, $\bar{\mathbf{x}}$ *is stable;*
- *if* $\frac{dV}{dt}(\mathbf{x}) < 0$ *for all* $\mathbf{x} \in E \setminus \{\bar{\mathbf{x}}\}$, $\bar{\mathbf{x}}$ *is asymptotically stable;*
- *if* $\frac{dV}{dt}(\mathbf{x}) > 0$ *for all* $\mathbf{x} \in E \setminus \{\bar{\mathbf{x}}\}$, $\bar{\mathbf{x}}$ *is unstable.*

Example 3.2. The goal of this example is to illustrate Theorem 3.1. Consider the system described by Equation (3.6).

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = -\mathbf{x} \tag{3.6}$$

This equation generates an attractive field towards the origin (see Figure 3.3), which is an equilibrium point: $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. Now, we need to

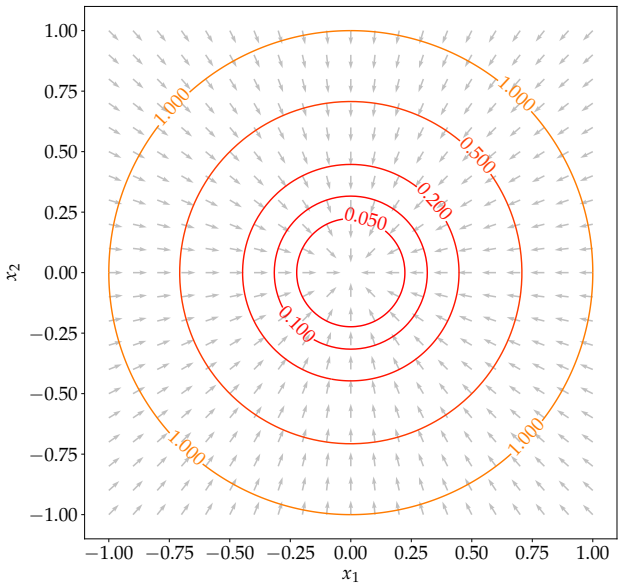


Figure 3.3: Attractive vector field of Equation (3.6) and contour lines of $V(\mathbf{x})$, in $\mathbb{R}^2$

find a function $V : \mathbb{R}^n \to \mathbb{R}$ such that $V(\mathbf{0}) = 0$ and $V(\mathbf{x}) > 0$ around the origin. Take for example

$$V(\mathbf{x}) = \|\mathbf{x}\|_2^2 = \sum_{i=1}^{n} x_i^2$$

which satisfies these properties. Let us now compute $\dot{V}(\mathbf{x})$:

$$\begin{aligned} \dot{V}(\mathbf{x}) &= 2 \cdot \mathbf{x}^{\mathrm{T}} \cdot \dot{\mathbf{x}} \\ &= -2 \cdot \mathbf{x}^{\mathrm{T}} \cdot \mathbf{x} \\ &< 0, \ \forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\} \end{aligned}$$

This proves asymptotic stability of the equilibrium point $\mathbf{0}$, according to Theorem 3.1.

Geometrically speaking, this theorem can be interpreted as follows: if the vector field defined by $\mathbf{f}$ is only incoming along a contour line of the function V, then the system is stable since it will never leave the neighbourhood enclosed by this surface level.

Remark 3.1. In the previous example, finding the Lyapunov function V was quite easy. In the case of a more complex system, it might not be as easy. In some cases, e. g. when systems represent a physical phenomenon, physical understanding of the system can be of help. Take the damped pendulum equation, for example. It has two equilibrium points: pendulum pointing upwards (unstable) and downwards (stable). A good candidate for a function decreasing as these equilibria are approached is the system's total energy.

Lyapunov's theory is the foundation of many methods related to stability of dynamical systems. For example, it is often used in control theory [51], [116] to prove stability of feedback loops and non-linear control methods.

3.2.1.3 *Periodic orbits & Poincaré maps*

3.2.1.3.1 ABOUT PERIODIC ORBITS

Oscillating systems can display stable behaviour when their oscillations tend towards the same pattern over time. Mathematically speaking, a system oscillates when for a given non-stable initial condition $\mathbf{x}_0$, its flow map $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ satisfies

$$\forall t \geq 0,\ \phi_{\mathbf{x}_0}(t+T) = \phi_{\mathbf{x}_0}(t)$$

where T is called the *period* of the oscillations. The set defined by $\gamma = \{\phi_{\mathbf{x}_0}(t), t \geq 0\}$ is then a closed trajectory, called a *closed orbit*. Note that an *orbit* of the system is not necessarily closed: an open orbit corresponds to a trajectory displaying a somewhat periodic pattern, that can asymptotically converge towards a closed orbit, diverge from it, or even display chaotic behaviour. A closed orbit can be stable, i. e. its neighbouring trajectories converge towards it as time tends to infinity, or unstable when the neighbouring trajectories are divergent. In those cases, the periodic orbit is usually called a *limit cycle*. Figure 3.4 illustrates the concepts mentioned above.

Usually, a limit cycle cannot be described by an analytical expression. However, numerical approaches based on set-membership methods can be used to find rigorous enclosures for limit cycles [62], [68].

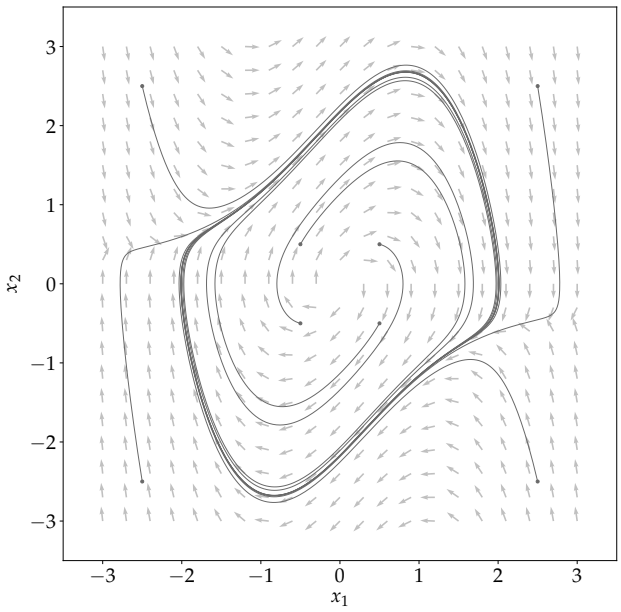


Figure 3.4: Approximating the limit cycle of the Van der Pol oscillator (Equation (2.3)). Neighbouring trajectories converge towards a limit cycle.

3.2.1.3.2 ABOUT POINCARÉ MAPS

Analysing stability of a limit cycle is however more complex than with an equilibrium point. The classical tool used to do so is the *Poincaré map* or *first recurrence map* [41, Chapter 10], [92, Chapter 3].

Definition 3.4 (Poincaré map)**.** Consider a system with a flow function $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, and a closed orbit γ of this system. Define a local section $\mathcal{S}$ crossing γ at a point $\mathbf{x}_0 \in \mathbb{R}^n$. Choose $\mathbf{x} \in \mathcal{S}$. Then the *Poincaré map* Π associated to γ is given by Equation (3.7), where $\tau_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{R}$ is the function returning the smallest positive time such that $\phi\left(\tau_{\mathcal{S}}\left(\mathbf{x}\right), \mathbf{x}\right) \in \mathcal{S}$.

$$\begin{aligned} \Pi : \mathcal{S} &\to \mathcal{S} \\ \mathbf{x} &\mapsto \phi\left(\tau_{\mathcal{S}}\left(\mathbf{x}\right), \mathbf{x}\right) \end{aligned} \tag{3.7}$$

The section $\mathcal{S}$ is called the *Poincaré section*, and is usually defined by a function $\sigma : \mathbb{R}^n \to \mathbb{R}$ such that $\sigma\left(\mathbf{x}\right) = 0$. It is an $(n\text{-}1)$-dimensional hypersurface of $\mathbb{R}^n$.

. . . provided that 0 is a regular value of σ

However, this definition restricts the Poincaré map to one surface $\mathcal{S}$, and the use of that tool to periodic orbits. A more general definition is sometimes used [45], [120] and called *local Poincaré map* [124], [125].

Definition 3.5 (Local Poincaré map)**.** Consider a system with a flow function $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, and a sequence of surfaces $(\mathcal{S}_k)_{k \in \mathbb{N}}$,

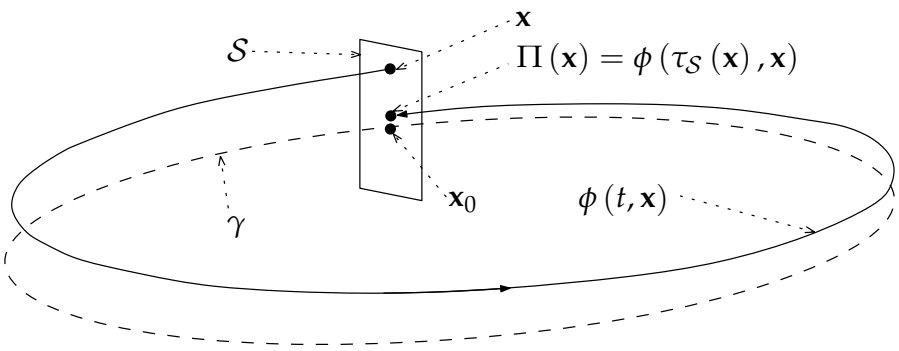


Figure 3.5: Illustration of the Poincaré map (Definition 3.4)

respectively transversal to the flow at points forming the sequence $(\mathbf{x}_k)_{k\in\mathbb{N}}$. Then the *local Poincaré map* Π_k is defined by Equation (3.8), where $\tau_{\mathcal{S}_k} : \mathbb{R}^n \to \mathbb{R}$ is the function returning the smallest positive time such that $\phi(\tau_{\mathcal{S}_k}(\mathbf{x}), \mathbf{x}) \in \mathcal{S}_k$.

$$\begin{aligned} \Pi_k : \mathcal{S}_k &\to \mathcal{S}_{k+1} \\ \mathbf{x}_k &\mapsto \phi\left(\tau_{\mathcal{S}_{k+1}}(\mathbf{x}_k), \mathbf{x}_k\right) \end{aligned} \tag{3.8}$$

This definition allows the use of Poincaré maps to non-periodic trajectories (see Figure 3.6) and broadens its possible uses.

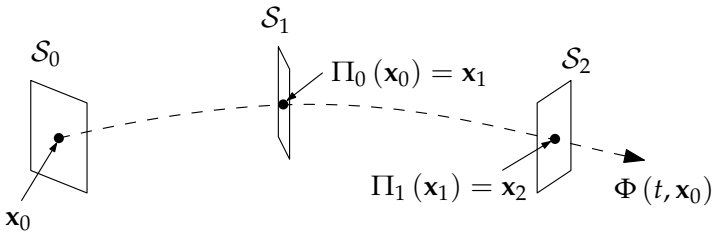


Figure 3.6: Illustration of the local Poincaré map (Definition 3.5)

In most cases, there is no analytical expression for a Poincaré map [125]. Usually, the differential equation describing the system is integrated over time until reaching the desired Poincaré section. The integration then stops, and the crossing point is returned as the result of the Poincaré map. Depending on the shape of the section, and whether it is orthogonal or not to the trajectory, different strategies must be used, like adjusting the integration step or using the trajectory's local derivative. More information can be found in [125], and insight on how Poincaré maps are computed in the CAPD library is available in [131].

3.2.1.3.3 COMPUTING THE DERIVATIVE OF A POINCARÉ MAP

Given that the function $\mathbf{f}$ describing the system is sufficiently continuous and differentiable, one can define the derivative of the Poincaré map associated with that system by Equation (3.9).

$$\begin{aligned}\frac{\partial \Pi}{\partial \mathbf{x}}(\mathbf{x}) &= \frac{\partial}{\partial \mathbf{x}}\left(\phi\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right)\right) \\ &= \frac{\partial \phi}{\partial \mathbf{x}}\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right) + \left(\frac{\partial \phi}{\partial t}\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right)\right)^{\mathrm{T}} \cdot \frac{\partial \tau_{\mathcal{S}}}{\partial \mathbf{x}}(\mathbf{x}) \\ &= \frac{\partial \phi}{\partial \mathbf{x}}\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right) + \left(\mathbf{f}\left(\phi\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right)\right)\right)^{\mathrm{T}} \cdot \frac{\partial \tau_{\mathcal{S}}}{\partial \mathbf{x}}(\mathbf{x}) \\ &= \frac{\partial \phi}{\partial \mathbf{x}}\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right) + \left(\mathbf{f}\left(\Pi(\mathbf{x})\right)\right)^{\mathrm{T}} \cdot \frac{\partial \tau_{\mathcal{S}}}{\partial \mathbf{x}}(\mathbf{x})\end{aligned} \tag{3.9}$$

Assuming that $\Pi(\mathbf{x})$ and $\tau_{\mathcal{S}}(\mathbf{x})$ are known, and considering that finding $\frac{\partial \phi}{\partial \mathbf{x}}\left(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}\right)$ comes down to finding the solution at time $\tau_{\mathcal{S}}(\mathbf{x})$ of the system's variational equation (see Paragraph 2.2.2.3.2), it only remains to compute the state-derivative of $\tau_{\mathcal{S}}(\mathbf{x})$.

Proposition 3.1. *Consider a system with a flow function $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ and a Poincaré section $\mathcal{S}$ defined by the function $\sigma : \mathbb{R}^n \to \mathbb{R}$. Let $\mathbf{x} \in \mathbb{R}^n$ and let $T = \tau_{\mathcal{S}}(\mathbf{x})$ denote its return time to the section $\mathcal{S}$ and define $\mathbf{y} = \phi(T, \mathbf{x}) = \Pi_{\mathbf{x}}$. Denote by $\mathbf{J}_{\phi_T}(\mathbf{x})$ the Jacobian matrix of the t-transition $\phi(T, \mathbf{x})$ and by $\nabla\sigma(\mathbf{y})$ the gradient of the section. Then, the space-derivative of $\tau_{\mathcal{S}}$ is given by*

$$\frac{\partial \tau_{\mathcal{S}}}{\partial \mathbf{x}}(\mathbf{x}) = -\frac{\nabla\sigma^{\mathrm{T}}(\mathbf{y})}{\nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot \mathbf{f}(\mathbf{y})} \cdot \mathbf{J}_{\phi_T}(\mathbf{x}) \tag{3.10}$$

Proof. We introduce the notations used below in Figure 3.7. One can notice that a small offset $d\mathbf{x}$ on the original state $\mathbf{x}$ will induce an offset $d\mathbf{y}$ of the impact point on the Poincaré surface $\mathcal{S}$.

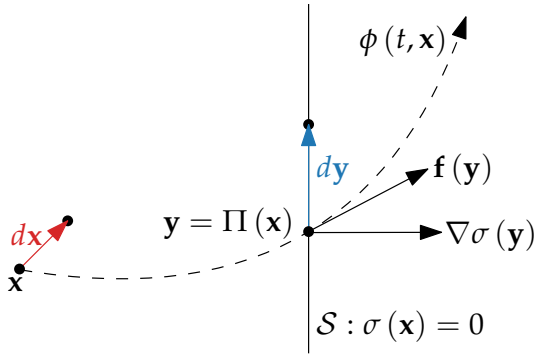


Figure 3.7: Computing the state derivative of the return time function

Since $d\mathbf{y}$ is orthogonal to $\nabla\sigma(\mathbf{y})$ for all $\mathbf{y} \in \mathcal{S}$, we have

$$\begin{aligned}
& \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot d\mathbf{y} = 0 \\
\Longrightarrow \quad & \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot d\Pi(\mathbf{x}) = 0 \\
\Longrightarrow \quad & \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot d\phi(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}) = 0 \\
\Longrightarrow \quad & \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot (d_t\phi(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x}) + d_{\mathbf{x}}\phi(\tau_{\mathcal{S}}(\mathbf{x}), \mathbf{x})) = 0 \\
\Longrightarrow \quad & \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot \left(d\tau_{\mathcal{S}}(\mathbf{x}) \cdot \frac{d\phi}{dt}(T, \mathbf{x}) + d_{\mathbf{x}}\phi(T, \mathbf{x}) \right) = 0 \\
\Longrightarrow \quad & \nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot (d\tau_{\mathcal{S}}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) + d_{\mathbf{x}}\phi(T, \mathbf{x})) = 0 \\
\Longrightarrow \quad & d\tau_{\mathcal{S}}(\mathbf{x}) = -\frac{\nabla\sigma^{\mathrm{T}}(\mathbf{y})}{\nabla\sigma^{\mathrm{T}}(\mathbf{y}) \cdot \mathbf{f}(\mathbf{y})} \cdot d_{\mathbf{x}}\phi(T, \mathbf{x}) \qquad (3.11)
\end{aligned}$$

Furthermore, by definition, we have

$$d\tau_{\mathcal{S}}(\mathbf{x}) = \frac{\partial\tau_{\mathcal{S}}}{\partial\mathbf{x}}(\mathbf{x}) \cdot d\mathbf{x} \qquad (3.12)$$

and

$$d_{\mathbf{x}}\phi(T, \mathbf{x}) = \frac{\partial\phi}{\partial\mathbf{x}}(T, \mathbf{x}) \cdot d\mathbf{x} = \mathbf{J}_{\phi_T}(\mathbf{x}) \cdot d\mathbf{x} \qquad (3.13)$$

Finally, combining Equations (3.11) to (3.13) yields Equation (3.10), which concludes this proof. ■

Example 3.3. This example will explain how to compute a Poincaré map and its first derivative for a simple pendulum system using CAPD.

Consider a simple pendulum such as described by Equation (2.41), with a damping coefficient $d = 5 \times 10^{-5}$. Let us define the section $\mathcal{S}$, described by Equation (3.14).

$$\begin{aligned}
& \sigma : \mathbb{R}^2 \to \mathbb{R} \\
& (\theta, \dot{\theta}) \mapsto \dot{\theta} - 0.5 \cdot \theta
\end{aligned} \qquad (3.14)$$

The following piece of code illustrates how CAPD can compute a Poincaré map and its first derivative.

3.2.1.3.4 ABOUT POINCARÉ MAPS AND STABILITY OF LIMIT CYCLES

The main purpose of the Poincaré map is to transform a possibly complex continuous-time system into a discrete one, which is much simpler to study. Indeed, the original continuous-time system given by Equation (3.1) can be transformed into the discrete-time system defined by Equation (3.15) by using Definition 3.4.

$$\mathbf{x}_{k+1} = \Pi(\mathbf{x}_k) \qquad (3.15)$$

```
1  void f(Node, Node x[], int, Node f[], int, Node params[], int)
     {
2    f[0] = x[1];
3    f[1] = -sin(x[0]) - 0.00005 * x[1];
4  }
5
6  int main(int argc, char** argv) {
7    int dimX = 2, dimF = 2, dimP = 0;
8    double epsilon = 1e-6;
9    capd::IMap vf(f, dimX, dimF, dimP);
10   capd::IOdeSolver solver(vf, 30);
11   capd::ITimeMap timeMap(solver);
12
13   capd::INonlinearSection section("var:x1,x2;fun:x2-0.5*x1;");
     // define the Poincaré section S
14   IPoincareMap pm(solver, section, capd::poincare::MinusPlus);
     // define the crossing direction of the section, in that case
      when s(x) changes from negative to positive
15
16   capd::IVector x0 = {{1 - epsilon, 1 + epsilon}, {0.5 -
     epsilon, 0.5 + epsilon}}; // Initial state of the pendulum
17   C1Rect2Set set(x0); // we chose a doubleton set to represent
     the solutions. Note the C1..., which allows computing the
     first derivative of the solution's trajectory
18
19   IMatrix monodromyMatrix(2, 2); // Matrix to store J_{φ_T}(x_0)
20   capd::interval returnTime; // Interval to store τ_S(x_0)
21
22   IVector x1 = pm(set, monodromyMatrix, returnTime); // x1 is
     the value of Π(x_0); monodromyMatrix and returnTime are set
23   IMatrix dP = pm.computeDP(x1, monodromyMatrix, returnTime);
     // dP corresponds to ∂Π/∂x (x_0)
24
25   return EXIT_SUCCESS;
26 }
27
```

Program 3.1: Computing a Poincaré map and its derivative with CAPD

In other words, the notion of time is suppressed (or, more precisely, dissimulated) using of Poincaré maps: its behaviour is now represented by discrete jumps in the state space, which can be regularly distributed in time (when the system evolves on a limit cycle for example) or not.

According to [119, Chapter 12], a limit cycle's stability can be proved using a Poincaré map.

Theorem 3.2. *Consider the continuous-time system described by Equation (3.1), with* $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$. *Denote by* γ *a limit cycle of the system. Let* Π *be a Poincaré map associated to a Poincaré section* $\mathcal{S}$ *crossing* γ *at a point*

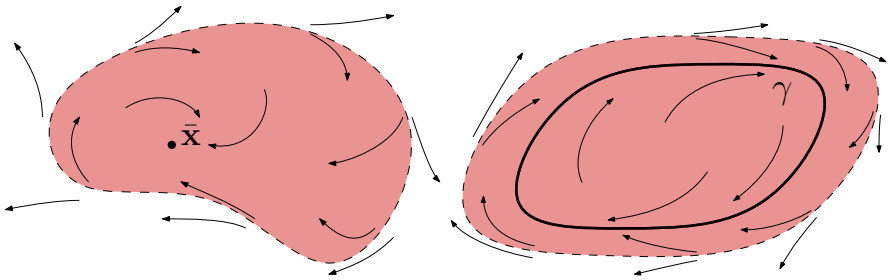
$\bar{\mathbf{x}}$. *Then,* γ *is (asymptotically) stable if and only if* $\bar{\mathbf{x}}$ *is an (asymptotically) stable equilibrium of* Π.

In other words, proving stability of a limit cycle comes down to proving stability of the discrete-time system associated to its Poincaré map (see Section 3.2.2).

Poincaré maps have been used to prove the existence of limit cycles of multiple systems, usually using set-membership methods for rigorousness [123], [132], [133]. Each time, a computer program was used to issue the proof (whence the use of interval computation methods).

3.2.1.4 *Attractors & basins of attraction*

Depending on the system's dynamics, the latter's trajectories can sometimes converge towards a stable point or a stable limit cycle. These mathematical objects are sometimes called *attractors*. Each attractor lies in an area where each trajectory tends towards it: its *basin of attraction* (see Figure 3.8).



(a) Stable equilibrium point $\bar{\mathbf{x}}$ (b) Stable limit cycle γ

Figure 3.8: Attractors $\bar{\mathbf{x}}$ and γ and their basin of attraction (in red)

Once again, finding an enclosure for a basin of attraction is no easy task. Set-membership methods are particularly suited for it [26], [71], [96], [109]. The resulting approximation of the basin of attraction is then a subpaving of non-overlapping boxes (see Section 2.6), the union of which either corresponds to the inner or the outer approximation of the actual basin.

3.2.2 *Stability of discrete dynamical systems*

The definitions of stability introduced in Section 3.2.1 are also valid for discrete-time systems [41, Chapter 16]. In their case, however, it is usually simpler to prove their stability, since they are defined by a recurrence map, not a differential equation requiring rigorous integration. When discussing discrete systems, the expressions *fixed point* and *periodic orbits* are generally used in place of equilibrium

points and limit cycles. Let us recall the general form of a discrete-time dynamical system

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \tag{3.16}$$

where $\mathbf{f} \in \mathcal{C}^m(\mathbb{R}^n)$.

Let us now introduce the Lyapunov's definitions of stability for discrete-systems.

Definition 3.6 (Stability)**.** The system described by Equation (3.16) is *stable* around the fixed point $\bar{\mathbf{x}}$ if

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \|\mathbf{x}_0 - \bar{\mathbf{x}}\| < \delta \implies \forall k \geq 0,\ \|\mathbf{x}_k - \bar{\mathbf{x}}\| < \varepsilon \tag{3.17}$$

Otherwise, it is *unstable*.

Definition 3.7 (Asymptotic stability)**.** The system is *asymptotically stable* around $\bar{\mathbf{x}}$ if it is *stable* and

$$\exists \delta > 0,\ \|\mathbf{x}_0 - \bar{\mathbf{x}}\| < \delta \implies \lim_{k \to \infty} \mathbf{x}_k = \bar{\mathbf{x}} \tag{3.18}$$

Definition 3.8 (Exponential stability)**.** The system is *exponentially stable* around $\bar{\mathbf{x}}$ if there exist $\alpha > 0$ and $\beta > 0$ such that

$$\exists \delta > 0,\ \|\mathbf{x}_0 - \bar{\mathbf{x}}\| < \delta \implies \|\mathbf{x}_k - \bar{\mathbf{x}}\| \leq \alpha \|\mathbf{x}_0 - \bar{\mathbf{x}}\| e^{-\beta \cdot k} \tag{3.19}$$

3.2.2.1 *Stability of fixed points*

To study stability of an equilibrium point $\bar{\mathbf{x}}$ of a continuous-time system could be achieved by checking its linearised counterpart's eigenvalues around $\bar{\mathbf{x}}$. This is still true for a discrete-time system.

Proposition 3.2. *Consider the system described by Equation (3.16), where $\mathbf{J}_\mathbf{f}(\bar{\mathbf{x}})$ is the Jacobian matrix of $\mathbf{f}$ evaluated at the fixed point $\bar{\mathbf{x}}$. Then,*

- *if the eigenvalues $(\lambda_i)_{i \in [1,n]}$ of $\mathbf{J}_\mathbf{f}(\bar{\mathbf{x}})$ are inside the unit circle, i. e. $|\lambda_i| < 1$, then $\bar{\mathbf{x}}$ is asymptotically stable for the system described by $\mathbf{f}$;*
- *if one of the eigenvalues $(\lambda_i)_{i \in [1,n]}$ of $\mathbf{J}_\mathbf{f}(\bar{\mathbf{x}})$ is outside the unit circle, then $\bar{\mathbf{x}}$ is unstable;*
- *otherwise, $\bar{\mathbf{x}}$ is either stable, unstable or neutral.*

Example 3.4. To illustrate this proposition, we will use the logistic map (already introduced in Example 2.5). We recall here its expression and compute its derivative.

$$x_{k+1} = f(x_k) = \rho \cdot x_k \cdot (1 - x_k) \tag{3.20}$$

$$f'(x_k) = \rho \cdot (1 - 2 \cdot x_k) \tag{3.21}$$

It is straightforward to find its fixed points:

$$\bar{x}_1 = 0$$
$$\bar{x}_2 = \frac{\rho - 1}{\rho}$$

Let us study their stability:

$$f'(\bar{x}_1) = \rho$$
$$f'(\bar{x}_2) = 2 - \rho$$

We can then conclude that $\bar{x}_1$ is asymptotically stable if $0 < \rho < 1$. Note that for those values of ρ, $\bar{x}_2$ is not "valid", since x_k is required to remain in the interval $[0, 1]$. We can also conclude that $\bar{x}_2$ is asymptotically stable if $1 < \rho < 3$.

For $\rho \geq 3$ the fixed points are not stable any more, but that does not mean that the system cannot converge towards another stable behaviour, e. g. a periodic orbit (see Example 3.5).

3.2.2.2 *Stability of periodic orbits*

A discrete-time system describing a periodic orbit evolves according to a fixed-size sequence of points:

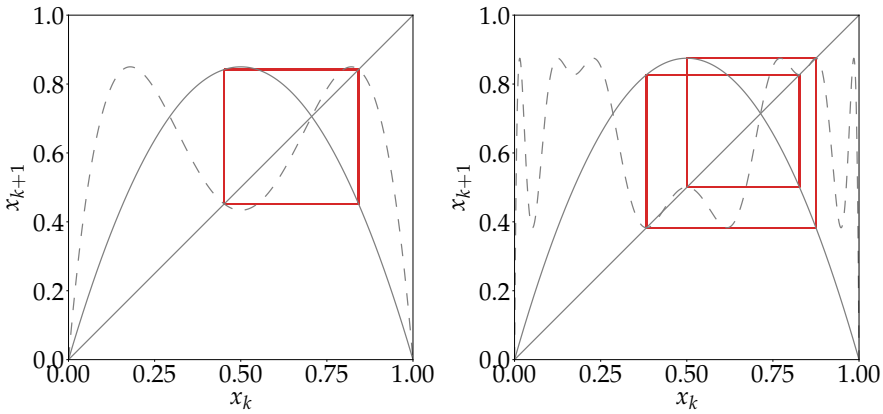
$$\mathbf{x}_0,\ \mathbf{x}_1,\ \ldots,\ \mathbf{x}_{N-1},\ \mathbf{x}_0,\ \mathbf{x}_1,\ \ldots,\ \mathbf{x}_{N-1},\ \mathbf{x}_0,\ \mathbf{x}_1,\ \ldots,\ \mathbf{x}_{N-1},\ \ldots$$

The point $\mathbf{x}_0$ is called the orbit's *seed*, and N corresponds to the latter's period. Now, since each point $\mathbf{x}_i$ is a fixed point of the system described by $\mathbf{f}^N$, proving stability of the periodic orbit comes down to proving stability of one $\mathbf{x}_i$, which can be done as explained in the previous section.

Example 3.5. Let us resume the development of Example 3.4, with $\rho \geq 3$. In that case, we start observing periodic orbits. For example, take $\rho = 3.4$: a periodic orbit of size 2 can be observed in Figure 2.4c. In other words, the sequence of $(x_k)_{k \in \mathbb{N}}$ converges towards a sequence alternating between two points $\bar{x}_3$ and $\bar{x}_4$. The latter are the roots of the following equation (fourth-order polynomial):

$$\bar{x} = f \circ f(\bar{x}) \tag{3.22}$$

Their values can be observed in Figure 3.9a. We can also observe graphically that the derivative of f^2 is negative for $\bar{x}_3$ and $\bar{x}_4$, meaning that the latter are stable fixed points for f^2, which proves stability of the periodic orbit. Similarly, for $\rho = 3.5$, an oscillating trajectory between four values can be observed (see Figure 3.9b).



(a) Graph of $f(x)$ (solid line), $f^2(x)$ (dashed line) in the plane (x_k, x_{k+1}) and resulting periodic orbit (in red)

(b) Graph of $f(x)$ (solid line), $f^4(x)$ (dashed line) in the plane (x_k, x_{k+1}) and resulting periodic orbit (in red)

Figure 3.9: Examples of periodic orbits of the logistic map

3.2.3 *Stability of hybrid systems*

In this section, we will discuss the stability of hybrid systems, the formalism of which we introduced in Section 2.2.4. In [34], [35], Girard gives an extensive introduction about stability of hybrid systems, upon which we based this section. This subject has also been addressed in earlier publications [42], [43], [113].

The main problem encountered in the literature is to prove stability of *hybrid limit cycles*. The latter are analogous to continuous ones or periodic orbits, to the difference that both the discrete and continuous parts of the system's state must describe a periodic pattern: $(q(t), \mathbf{x}(t)) = (q(t+T), \mathbf{x}(t+T))$ where T is the period of the system. In the rest of this chapter, we will consider that the studied hybrid systems have limit cycles.

The principle of the methods developed in [34], [35] is based on the use of the Poincaré map (see Paragraph 3.2.1.3.2). Consider a hybrid system $\mathcal{H} = (\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{F}, \mathcal{G}, \mathcal{R})$ and an initial state $(q_0, \mathbf{x}_0)$. Each guard $G_{e_{ij}}$ of the system, which delimits the domains D_{q_i} and D_{q_j}, can be interpreted as a Poincaré section. That is, for each $G_{e_{ij}}$, a local Poincaré map Π_{q_i} associated with the vector field $\mathbf{f}_{q_i}$ can be defined as follows:

$$\begin{aligned} \Pi_{q_i} : D_{q_i} &\to G_{e_{ij}} \\ \mathbf{x} &\mapsto \phi_{q_i}\left(\tau_{G_{e_{ij}}}(\mathbf{x}), \mathbf{x}\right) \end{aligned} \tag{3.23}$$

Therefore, the continuous state of the system can be described by a sequence of discrete points $(\mathbf{x}_k)_{k\in\mathbb{N}}$, as given by Equation (3.24) if we assume that the transitions between the discrete states $q_i \in \mathcal{Q}$ are

executed in the order of increasing k. N corresponds to the number of discrete states.

$$\begin{aligned}\mathbf{x}_{N+1} &= \Pi_{q_N} \circ \cdots \circ \Pi_{q_0}(\mathbf{x}_0) \\ &= \Pi(\mathbf{x}_0)\end{aligned} \tag{3.24}$$

Π is called the *hybrid Poincaré map* of $\mathcal{H}$ [34], and is illustrated in figure 3.10.

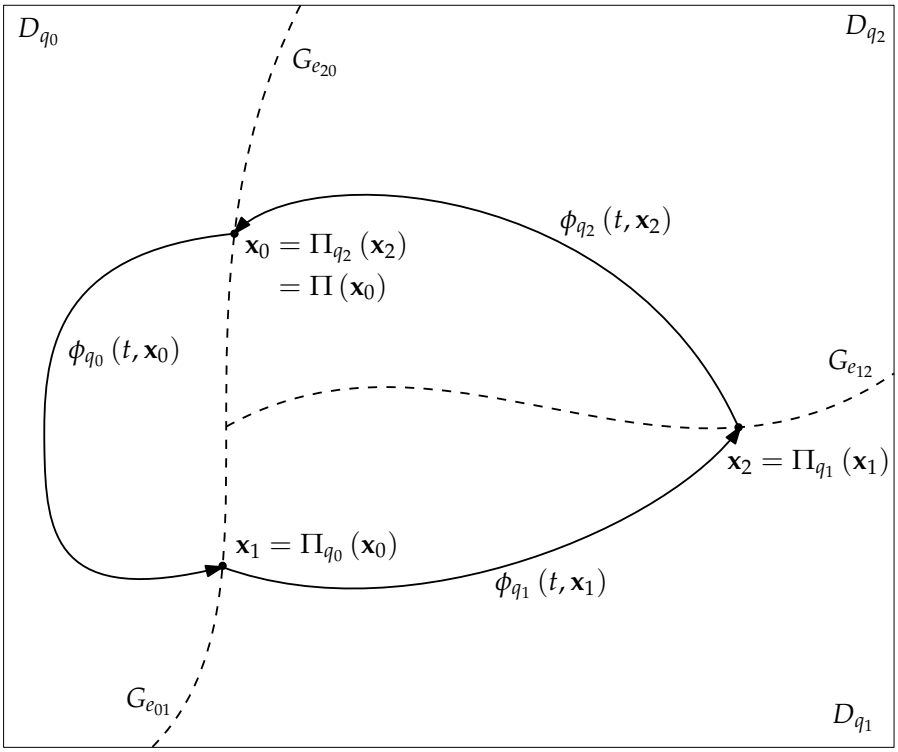


Figure 3.10: Graphical representation of the hybrid Poincaré map on a limit cycle of the system

Remark 3.2. Note that since $\Pi_{q_i}(\mathbf{x}) \in G_{e_{ij}}$, the associated reset function $R_{e_{ij}}$ is triggered by the Poincaré map. Therefore, each Poincaré map Π_{q_i} maps the reset state $R_{e_{ij}}(\mathbf{x})$. For the sake of clarity, we will not make this appear in the equations later.

Considering that each vector field $\mathbf{f}_{q_i}$ is continuous and smooth enough, the derivative of Π with respect to $\mathbf{x}$ is expressed as follows:

$$\frac{\partial \Pi}{\partial \mathbf{x}}(\mathbf{x}) = \prod_{i=N}^{0} \frac{\partial \Pi_{q_i}}{\partial \mathbf{x}} \left(\Pi_{q_{i-1}} \circ \cdots \circ \Pi_{q_0}(\mathbf{x}_0) \right) \tag{3.25}$$

where $\frac{\partial \Pi_{q_i}}{\partial \mathbf{x}}$ corresponds to the derivative of a local Poincaré map (see Equation (3.9)).

Finally, to conclude about stability of $\mathcal{H}$, one simply needs to apply Proposition 3.2, i. e. check that the eigenvalues of $\frac{\partial \Pi}{\partial \mathbf{x}}$ are inside the unit circle.

For example, Girard has applied this method to prove stability of the limit cycles of piecewise-linear hybrid systems in [34], [42].

3.3 FORMALISATION OF A DOCKING MISSION AS A STABILITY PROBLEM

In this section, we will formalise a docking mission as a stability problem. We will then highlight the different challenges inherent to this formalisation and the various tools available to solve the problem.

As explained earlier in this thesis, a robot can be modelled as a hybrid system: the physical phenomenons acting on the robot and the thrusts of its propellers can be modelled as a continuous system, and its sensors, localisation and control algorithms as a discrete system. Denote by $\mathbf{x}(t) \in \mathbb{R}^n$ the continuous state (e. g. the position, orientation, linear and angular velocities...), by $\hat{\mathbf{x}} \in (t)\,\mathbb{R}^m$ the state estimation, contained inside the robot's computer, by $\mathbf{u}(t)$ the command issued by the controller, and by $\mathbf{y}(t)$ the vector of measurements. Denote by $\delta > 0$ the update period of the computer, and a sequence of update times $(t_k)_{k\in\mathbb{N}}$ such that $t_k = k\delta$.

Since the state estimation, the commands issued to the motors and the measurements are processed by the computer, we have

$$\forall k > 0,\ \forall t \in [t_k, t_{k+1}],\ \begin{cases} \hat{\mathbf{x}}(t) = \hat{\mathbf{x}}(t_k) = \hat{\mathbf{x}}_k \\ \mathbf{u}(t) = \mathbf{u}(t_k) = \mathbf{u}_k \\ \mathbf{y}(t) = \mathbf{y}(t_k) = \mathbf{y}_k \end{cases} \tag{3.26}$$

The complete model of the robot is

$$\forall k > 0,\ \begin{cases} \dot{\mathbf{x}}(t) = \boldsymbol{\varphi}(\mathbf{x}(t), \mathbf{u}_k),\ \forall t \in [t_k, t_{k+1}] \\ \hat{\mathbf{x}}_{k+1} = \hat{\boldsymbol{\varphi}}(\hat{\mathbf{x}}_k, \mathbf{y}_k, \mathbf{u}_k) \\ \mathbf{u}_k = \boldsymbol{\gamma}(\hat{\mathbf{x}}_k) \\ \mathbf{y}_k = \boldsymbol{\zeta}(\mathbf{x}(t_k)) \end{cases} \tag{3.27}$$

Equation (3.27) can be reformulated into

$$\forall k > 0,\ \begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \hat{\mathbf{x}}_k),\ \forall t \in [t_k, t_{k+1}] \\ \hat{\mathbf{x}}_{k+1} = \hat{\mathbf{f}}(\hat{\mathbf{x}}_k, \mathbf{x}(t_k)) \end{cases} \tag{3.28}$$

Concerning the target of the docking mission, we assume that its state is somehow known by the robot's computer. For example, if the target is moving (e. g. an USV onto which the robot must dock), this assumption implies that its state $\bar{\mathbf{x}}(t_k)$ is sent periodically to the robot. The target's state can then be used as a variable of the controller, and combined inside Equation (3.28). If the target is fixed (e. g. a garage

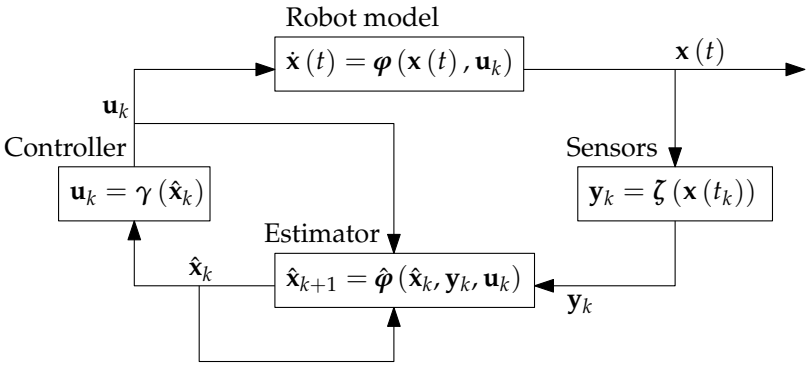


Figure 3.11: Representation of the robot's model

placed on the sea bottom), its position must be communicated to the robot prior to the mission, and will then be used as a parameter inside the controller. To sum up, Equation (3.28) is a model for the robot and its target, and is sufficient to determine the robot's behaviour over time.

We consider one single discrete state q, because we consider that the laws describing the robot's behaviour are the same no matter its position, orientation, etc... Furthermore, since $\hat{\mathbf{x}}$ is constant over a time step, we have $\dot{\hat{\mathbf{x}}} = \mathbf{0}_m$

Now, the system described by Equation (3.28) can be formalised as a hybrid system, exactly like explained in Example 2.8. The robot only has one discrete state q, its continuous state being the vector $\mathbf{z} \in \mathbb{R}^{n+m+1}$ composed of $\mathbf{x}(t)$, $\hat{\mathbf{x}}(t_k)$ and of a time variable τ. Its behaviour is defined by

$$\forall k > 0,\ \forall t \in [t_k, t_{k+1}],\ \dot{\mathbf{z}} = \begin{pmatrix} \mathbf{f}(\mathbf{x}(t), \hat{\mathbf{x}}_k) \\ \mathbf{0}_m \\ 1 \end{pmatrix} \tag{3.29}$$

The only guard of the system $G_{(q,q)}$ is defined by

$$G_{(q,q)} = \left\{ \mathbf{z} \in \mathbb{R}^{n+m+1}, \tau = \delta \right\} \tag{3.30}$$

and triggers the reset function defined by

$$R_{(q,q)} = \left(\mathbf{x}(t), \hat{\mathbf{f}}(\hat{\mathbf{x}}_k, \mathbf{x}(t_k)), 0 \right) \tag{3.31}$$

If the localisation and control algorithms are well implemented, and the robot correctly modelled, the latter should reach the target state $\bar{\mathbf{x}}_k$. In other words, both $\mathbf{x}(t)$ and $\mathbf{x}_k$ should converge towards $\bar{\mathbf{x}}_k$. Now, one can notice that the variable τ is periodic, by definition. Therefore, $G_{(q,q)}$ is a Poincaré section, crossed every τ seconds by the system. Thus, proving that the robot will converge towards its target comes down to proving that its hybrid Poincaré map converges towards a stable hybrid cycle.

Now, the different models introduced earlier, their parameters and their states can be uncertain. Furthermore, they are most certainly

non-linear. Therefore, we need a method able to prove stability of a hybrid limit cycle described by a non-linear uncertain hybrid system. As explained in the previous section, this comes down to proving stability of a discrete system (defined by the hybrid Poincaré map). To the extent of our knowledge, such a method does not exist. Moreover, it is likely that the controllers and the localisation method might be working, i. e. driving the system towards its target, only in a given area of the state space. Knowing that the target is attracting the robot is therefore not sufficient for the robot's user, who also needs information regarding the "safe" zones inside which the robot can be initialised to reach its target. To the extent of our knowledge, a method able to provide a neighbourhood inside which a system is stable does not exist either.

These are the challenges we tackle in the rest of this chapter.

3.4 PROVING STABILITY OF UNCERTAIN DYNAMICAL SYSTEMS

Now that we have introduced the basics of stability of dynamical systems, we will focus on this thesis's first contributions. We propose here a general method for proving stability of uncertain discrete, continuous and hybrid systems. Existing methods tackle these problems by checking the eigenvalues of the linearised system [49], by approximating the system using piecewise linear functions [34] or by considering only linear systems [101]. Unlike some of the latter, our method is not based on a Lyapunov function, which simplifies the problem for complex dynamical systems. It can be applied to non-linear systems and can deal with uncertainties. Finally, it allows finding a value for the neighbourhoods ε and δ of the Lyapunov's definitions of stability, which is not achievable by any other existing method for complex non-linear systems, to the extent of our knowledge.

Our method is based on two major concepts: the *stability contractor*, that we introduce in Section 3.4.1, and iterative algorithms to compute the centred form of a composition of functions, presented in Section 3.4.2.

3.4.1 *Stability contractor*

We briefly introduced the concept of contractor in Section 2.5, which is an operator that contracts a set according to a set of constraints. Here, we adapted this tool to create a *stability contractor*, i. e. an operator that can contract a box in a way that can imply asymptotic stability of the system which state is enclosed by that box. Note that we initially introduced this tool in [10].

Definition 3.9. Let $[\mathbf{x}_0] \ni \mathbf{0}$ be a box of $\mathbb{R}^n$, and choose any $[\mathbf{a}] \ni \mathbf{0}$ and $[\mathbf{b}] \ni \mathbf{0}$ inside $[\mathbf{x}_0]$. A *stability contractor* of rate $\alpha < 1$ is an operator $\Psi : \mathbb{IR}^n \to \mathbb{IR}^n$ which satisfies

$$[\mathbf{a}] \subset [\mathbf{b}] \implies \Psi([\mathbf{a}]) \subset \Psi([\mathbf{b}]) \quad \text{(monotonicity)} \tag{3.32}$$

$$\Psi([\mathbf{a}]) \subset [\mathbf{a}] \quad \text{(contractance)} \tag{3.33}$$

$$\Psi(\mathbf{0}) = \mathbf{0} \quad \text{(equilibrium)} \tag{3.34}$$

$$\Psi([\mathbf{a}]) \subset \alpha \cdot [\mathbf{a}] \implies \forall k \geq 1,\ \Psi^k([\mathbf{a}]) \subset \alpha^k \cdot [\mathbf{a}] \quad \text{(convergence)} \tag{3.35}$$

where for $k \geq 1$, Ψ^k denotes the operator composition $\underbrace{\Psi \circ \cdots \circ \Psi}_{k}$, and Ψ^0 the identity function.

Example 3.6. Consider the operators given by Equation (3.36) and Equation (3.37).

$$\begin{aligned} \Psi_1 : \mathbb{IR} &\to \mathbb{IR} \\ [x] &\mapsto [x] \cap -0.9\,[x] \end{aligned} \tag{3.36}$$

$$\begin{aligned} \Psi_2 : \mathbb{IR} &\to \mathbb{IR} \\ [x] &\mapsto -0.9\,[x] \end{aligned} \tag{3.37}$$

Ψ_1 is a stability contractor, while Ψ_2 is not (the contractance property is not verified, take for example $[x] = [-1, 2]$, then $\Psi_2([x]) = [-1.8, 0.9] \not\subset [x]$).

Proposition 3.3. *If Ψ is a stability contractor of rate $\alpha < 1$, then*

$$\Psi([\mathbf{x}]) \subset \alpha \cdot [\mathbf{x}] \implies \lim_{k \to \infty} \Psi^k([\mathbf{x}]) = \mathbf{0} \tag{3.38}$$

Proof. Let $[\mathbf{x}] \in \mathbb{IR}^n$ and Ψ be a stability contractor of rate $\alpha < 1$. It follows that

$$\begin{aligned} \Psi([\mathbf{x}]) \subset \alpha \cdot [\mathbf{x}] &\implies \Psi^k([\mathbf{x}]) \subset \alpha^k \cdot [\mathbf{x}] \\ &\qquad \text{according to Equation (3.35)} \\ &\implies \lim_{k \to \infty} \Psi^k([\mathbf{x}]) = \mathbf{0} \quad \text{since } \alpha^k \to \mathbf{0} \text{ as } k \to \infty \end{aligned}$$

■

Corollary 3.1. *Consider a discrete-time system described by $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$ such that $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. Denote by $[\mathbf{x}_0] \ni \mathbf{0}$ the enclosure of the initial state $\mathbf{x}_0$, and let Ψ be a stability contractor of rate $\alpha < 1$ such that $\Psi([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$. If for all $k \geq 0$, $\mathbf{f}^k([\mathbf{x}_0]) \subseteq \Psi^k([\mathbf{x}_0])$, then the system is* asymptotically *stable in the neighbourhood $[\mathbf{x}_0]$.*

Proof. Let Ψ be a stability contractor of rate $\alpha < 1$ such that $\Psi([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$. Assume that for all $k \geq 0$, $\mathbf{f}^k[\mathbf{x}_0] \subseteq \Psi^k([\mathbf{x}_0])$. Then, according to Proposition 3.3 it follows that

$$\begin{aligned}
\Psi([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0] &\implies \lim_{k\to\infty} \Psi^k([\mathbf{x}_0]) = \mathbf{0} \\
&\implies \lim_{k\to\infty} \mathbf{f}^k([\mathbf{x}_0]) = \mathbf{0} \\
&\qquad\qquad \text{since } \forall k \geq 0,\ \mathbf{f}^k([\mathbf{x}_0]) \subseteq \Psi^k([\mathbf{x}_0]) \\
&\implies \forall \mathbf{x}_0 \in [\mathbf{x}_0],\ \lim_{k\to\infty} \mathbf{f}^k(\mathbf{x}_0) = \mathbf{0} \\
&\implies \forall \mathbf{x}_0 \in [\mathbf{x}_0],\ \lim_{k\to\infty} \mathbf{x}_k = \mathbf{0}
\end{aligned}$$

This corresponds to the definition of asymptotic stability (see Definition 3.7), which means that each $\mathbf{x}_0 \in [\mathbf{x}_0]$ will converge asymptotically towards the fixed point $\mathbf{0}$. ■

Remark 3.3. Note that this corollary allows proving stability for discrete systems having $\mathbf{0}$ as a fixed point and for neighbourhoods $[\mathbf{x}_0] \ni \mathbf{0}$. If the fixed point is non zero, i. e. $\mathbf{f}(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$, $\bar{\mathbf{x}} \neq \mathbf{0}$, $[\mathbf{x}_0] \ni \bar{\mathbf{x}}$, then one simply needs to centre the system on $\bar{\mathbf{x}}$ as follows:

- Choose $[\mathbf{z}_0] = [\mathbf{x}_0] - \bar{\mathbf{x}}$, then $\mathbf{0} \in [\mathbf{z}_0]$
- Consider the function $\mathbf{g}(\mathbf{z}) = \mathbf{f}(\mathbf{z} + \bar{\mathbf{x}}) - \mathbf{f}(\bar{\mathbf{x}})$, then $\mathbf{g}(\mathbf{0}) = \mathbf{0}$

Then the system $\mathbf{z}_{k+1} = \mathbf{g}(\mathbf{z}_k)$ and the initial box $[\mathbf{z}_0]$ meet the requirements of Corollary 3.1.

The following proposition is almost identical to Corollary 3.1, but it implies exponential – instead of asymptotic – stability.

Proposition 3.4. *Consider a discrete-time system described by* $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$ *such that* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. *Denote by* $[\mathbf{x}_0] \ni \mathbf{0}$ *the enclosure of the initial state* $\mathbf{x}_0$, *and let* Ψ *be a stability contractor of rate* $\alpha < 1$ *such that* $\Psi([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$. *If for all* $k \geq 0$, $\mathbf{f}^k([\mathbf{x}_0]) \subseteq \Psi^k([\mathbf{x}_0])$, *then the system is* exponentially *stable in the neighbourhood* $[\mathbf{x}_0]$.

Proof. Take any point $\mathbf{x}_0 \in [\mathbf{x}_0]$ and denote by $[\mathbf{x}'_0]$ the smallest box containing both $\mathbf{x}_0$ and $\mathbf{0}$. We have $[\mathbf{x}'_0] \subset [\mathbf{x}_0]$. Then, according to Equation (3.32),

$$\begin{aligned}
[\mathbf{x}'_0] \subset [\mathbf{x}_0] &\implies \Psi([\mathbf{x}'_0]) \subset \Psi([\mathbf{x}_0]) \subset \alpha[\mathbf{x}_0] \\
&\implies \exists \alpha_0 \leq \alpha,\ \Psi([\mathbf{x}'_0]) \subset \alpha_0[\mathbf{x}'_0] \\
&\implies \forall k \geq 1,\ \Psi^k([\mathbf{x}'_0]) \subset \alpha_0^k[\mathbf{x}'_0]
\end{aligned}$$

Define $\left[\mathbf{x}'_{k+1}\right] = \Psi^k\left(\left[\mathbf{x}'_0\right]\right)$ for all $k \geq 1$. Then, according to Equation (3.35) and Lemma 2.1,

$$\begin{aligned}\forall k \geq 1,\ \left[\mathbf{x}'_{k+1}\right] \subset \alpha^k \cdot \left[\mathbf{x}'_0\right] &\implies \| \left[\mathbf{x}'_{k+1}\right] \| < \alpha_0^k \cdot \| \left[\mathbf{x}'_0\right] \| \\ &\implies \| \left[\mathbf{x}'_{k+1}\right] \| < \| \left[\mathbf{x}'_0\right] \| \cdot e^{k \ln(\alpha_0)} \\ &\implies \| \left[\mathbf{x}'_{k+1}\right] \| < \| \left[\mathbf{x}'_0\right] \| \cdot e^{-k\beta}\end{aligned}$$

where $\beta = -\ln(\alpha_0) > 0$.

Since $\mathbf{x}_0$ is the furthest point from the origin inside $[\mathbf{x}'_0]$ by construction, we have $\| [\mathbf{x}'_0] \| \leq \|\mathbf{x}_0\|$.

Moreover, define $\mathbf{x}_k = \mathbf{f}^k(\mathbf{x}_0)$. We have $\mathbf{x}_k \in \left[\mathbf{x}'_k\right]$, and in turn $\|\mathbf{x}'_k\| \leq \| \left[\mathbf{x}'_k\right] \|$. It follows that

$$\forall k \geq 1,\ \|\mathbf{x}_k\| < \|\mathbf{x}_0\| \cdot e^{-k\beta}$$

Since this is true for all $\mathbf{x}_0 \in [\mathbf{x}_0]$, this implies exponential stability of the system around the origin, according to Definition 3.8. ■

Now, we have shown that our method can prove stability of a discrete system. However, one could legitimately ask whether our method can prove stability for any system as long as the latter is stable. Proposition 3.5 states that if the system is exponentially stable, this automatically implies $[\mathbf{f}_c]([\mathbf{x}_0]) \in [\mathbf{x}_0]$, for any $[\mathbf{x}_0]$ chosen carefully, and thus that our method can always prove stability of a stable system.

Proposition 3.5. *Consider a discrete-time system described by* $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$ *such that* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$*, and assume that it is exponentially stable around* $\mathbf{0}$*. Then there exists* $\eta > 0$ *and* $\gamma < 1$ *such that*

$$\forall [\mathbf{x}] \in \mathcal{X}_\eta,\ \exists k > 0,\ \left[\mathbf{f}_c^k\right]([\mathbf{x}]) \subset \gamma \cdot [\mathbf{x}] \tag{3.39}$$

where $\mathcal{X}_\eta = \{[\mathbf{x}] \ni \mathbf{0},\ \| [\mathbf{x}] \| \leq \eta\}$.

Proof. Assume that the system described by $\mathbf{f}$ is exponentially stable around the fixed point $\mathbf{0}$. Then, Definition 3.8 states that there exists $\alpha > 0$ and $\beta > 0$ such that

$$\exists \delta > 0,\ \|\mathbf{x}_0\| < \delta \implies \|\mathbf{x}_k\| < \alpha \|\mathbf{x}_0\| e^{-\beta \cdot k}$$

where $\mathbf{x}_0$ is a point in a neighbourhood δ of $\mathbf{0}$, and $\mathbf{x}_k = \mathbf{f}^k(\mathbf{x}_0)$. Note that for all $k \geq 0$, $\alpha e^{-\beta \cdot k} \leq \alpha$.

Define a box $[\mathbf{x}_0] \ni \mathbf{0}$ such that $\| [\mathbf{x}_0] \| < \delta$. Then,

$$\begin{aligned}\forall \mathbf{x}_0 \in [\mathbf{x}_0],\ \|\mathbf{x}_0\| < \delta &\implies \|\mathbf{f}^k(\mathbf{x}_0)\| < \alpha \|\mathbf{x}_0\| \\ &\implies \left[\mathbf{f}^k([\mathbf{x}_0])\right] \subset \alpha\,[\mathbf{x}_0]\end{aligned}$$

where $\left[\mathbf{f}^k\left([\mathbf{x}_0]\right)\right]$ is the minimal image of $[\mathbf{x}_0]$ by $\mathbf{f}^k$.

Now take any box $[\mathbf{y}] \ni \mathbf{0}$ in $[\mathbf{x}_0]$ and define $\eta = \|[\mathbf{y}]\|$. Then, by choosing η sufficiently small, the pessimism of the centred form becomes arbitrarily small and we can find γ such that $\alpha \leq \gamma < 1$ that verifies

$$\left[\mathbf{f}_c^k\right]([\mathbf{y}]) \subset \gamma\,[\mathbf{y}]$$

Finally, let us define $\mathcal{X}_\eta = \{[\mathbf{x}] \ni \mathbf{0},\ \|[\mathbf{x}]\| \leq \eta\}$. We then have

$$\forall\,[\mathbf{x}] \in \mathcal{X}_\eta,\ \exists k > 0,\ \left[\mathbf{f}_c^k\right]([\mathbf{x}]) \subset \gamma \cdot [\mathbf{x}]$$

■

3.4.2 *Proving stability of discrete-time systems*

Thanks to the stability contractor, one can prove stability of discrete-time dynamical systems. We will now explain how to build a stability contractor for a specific system. To achieve that, our stability contractor must meet the requirements of both Definition 3.9 and Corollary 3.1. In Section 2.3.4, we presented the centred form of an interval function, which is an inclusion function, i. e. it encloses the image set of an interval by a function. We will show in the following paragraphs that we can build a stability contractor using the centred form. We will also provide algorithms to compute the centred form of composition of functions. We introduced the latter in [10], [11].

In the rest of this section, we assume that the functions $\mathbf{f}$ describing the studied discrete-time systems are at least $\mathcal{C}^1$.

3.4.2.1 *Discrete iterative centred form centred on the origin*

Consider a discrete-time system described by $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$ such that $\mathbf{f}(\mathbf{0}) = \mathbf{0}$, and denote by $[\mathbf{x}_0]$ the enclosure of its initial state. Then the successive image sets $\mathbf{f}^k([\mathbf{x}_0])$ are enclosed by the centred form $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0])$. We propose below an iterative algorithm to compute $\left[\mathbf{f}_c^k\right]$.

Theorem 3.3. *Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{0}) = \mathbf{0}$ and denote by $\mathbf{J}_\mathbf{f}$ its Jacobian matrix. Let $[\mathbf{x}_0] \ni \mathbf{0}$ be a box of $\mathbb{IR}^n$. The centred form $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0])$ enclosing $\mathbf{f}^k([\mathbf{x}_0])$ is given by the following sequence*

$$\begin{aligned}
[\mathbf{z}_0] &= [\mathbf{x}_0] \\
[\mathbf{A}_0] &= \mathbf{I}_n \\
[\mathbf{z}_{k+1}] &= [\mathbf{f}]([\mathbf{z}_k]) \\
[\mathbf{A}_{k+1}] &= [\mathbf{J}_\mathbf{f}]([\mathbf{z}_k]) \cdot [\mathbf{A}_k] \\
\left[\mathbf{f}_c^k\right]([\mathbf{x}_0]) &= [\mathbf{A}_k] \cdot [\mathbf{x}_0]
\end{aligned} \tag{3.40}$$

Proof. Let us define $[\mathbf{z}_0] = [\mathbf{x}_0]$ and $[\mathbf{A}_0] = \mathbf{I}_n$.

For all $\mathbf{x} \in \mathbb{R}^n$, the Jacobian matrix of $\mathbf{f}^k$ is given by

$$\begin{aligned}\frac{d\mathbf{f}^k}{d\mathbf{x}}(\mathbf{x}) &= \frac{d(\mathbf{f} \circ \mathbf{f}^{k-1})}{d\mathbf{x}}(\mathbf{x}) \\ &= \frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{f}^{k-1}(\mathbf{x})) \cdot \frac{d\mathbf{f}^{k-1}}{d\mathbf{x}}(\mathbf{x})\end{aligned}$$

Equation (3.41) is the discrete version of the variational equation of a continuous-time system.

Define $\mathbf{z}_k = \mathbf{f}^k(\mathbf{x})$ and $\mathbf{A}_k = \frac{d\mathbf{f}^k}{d\mathbf{x}}(\mathbf{x})$. Since for all $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{J}(\mathbf{x}) = \frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{x})$, we have

$$\mathbf{A}_k = \mathbf{J}\left(\mathbf{z}_{k-1}\right) \cdot \mathbf{A}_{k-1} \tag{3.41}$$

Now, since $\mathbf{f}\left(\mathbf{0}\right) = \mathbf{0}$, it follows that $\mathbf{f}^k\left(\mathbf{0}\right) = \mathbf{0}$ for all k. Therefore, according to Definition 2.8 and Equation (3.41), it follows that

$$\begin{aligned}\left[\mathbf{f}_c^k\right]\left([\mathbf{x}_0]\right) &= \left[\mathbf{J}_{\mathbf{f}^k}\right]\left([\mathbf{x}_0]\right) \cdot [\mathbf{x}_0] \\ &= \left[\frac{d\mathbf{f}^k}{d\mathbf{x}}\right]\left([\mathbf{x}_0]\right) \cdot [\mathbf{x}_0] \\ &= [\mathbf{A}_k] \cdot [\mathbf{x}_0]\end{aligned} \tag{3.42}$$

■

Remark 3.4. Theorem 3.3 proposes an algorithm capable of computing enclosures for the successive states of a discrete system. Thanks to the use of the centred form, this algorithm should yield good results in terms of wrapping effect, as long as the initial box $[\mathbf{x}_0]$ is small enough (see Section 2.3.4 and Equation (2.33) in particular). However, this algorithm is not as performing as a Lohner-like algorithm for discrete systems in terms of sharpness of the enclosure. Actually, when k gets too large, the size of $\left[\mathbf{f}_c^k\right]\left([\mathbf{x}_0]\right)$ tends to diverge.

In its iterative form, the centred form is thus a tool allowing to enclose the trajectory of a discrete system. We will now show that under certain conditions, it becomes a stability contractor.

Theorem 3.4. *Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. If there exists a box $[\mathbf{x}_0] \ni \mathbf{0}$ and a real $\alpha < 1$ such that $[\mathbf{f}_c]\left([\mathbf{x}_0]\right) \subset \alpha \cdot [\mathbf{x}_0]$, then the operator Ψ defined by*

$$\begin{aligned}\Psi : \mathbb{IR}^n &\to \mathbb{IR}^n \\ [\mathbf{x}] &\mapsto [\mathbf{f}_c]\left([\mathbf{x}]\right)\end{aligned} \tag{3.43}$$

is a stability contractor inside $[\mathbf{x}_0]$.

Proof. The goal here is to prove that the centred form possesses the properties from Definition 3.9. Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a function such

that $\mathbf{f}(\mathbf{0}) = \mathbf{0}$, and define a box $[\mathbf{x}_0] \ni \mathbf{0}$ and a real $\alpha < 1$ such that $[\mathbf{f}_c]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$.

The monotonicity (Equation (3.32)) of Ψ results from that of the natural inclusion function of $\mathbf{J_f}$.

The contractance (Equation (3.33)) results from the original assumption that $[\mathbf{f}_c]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$.

The equilibrium (Equation (3.34)) is straightforward: $[\mathbf{f}_c](\mathbf{0}) = [\mathbf{J_f}](\mathbf{0}) \cdot \mathbf{0} = \mathbf{0}$

We will demonstrate the convergence property (Equation (3.35)) by induction. We want to prove that

$$[\mathbf{f}_c]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0] \implies \forall k \geq 0,\ [\mathbf{f}_c]^k([\mathbf{x}_0]) \subset \alpha^k \cdot [\mathbf{x}_0] \tag{3.44}$$

Note that $[\mathbf{f}_c]^k([\mathbf{x}_0])$ is not the same quantity than $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0])$. The first one refers to the composition of the centred form of $\mathbf{f}$, while the second one refers to the centred form of $\mathbf{f}^k$.

Equation (3.44) holds for $k = 0$: $[\mathbf{x}_0] \subset \alpha^0 \cdot [\mathbf{x}_0]$. Let us define the sequence

$$[\mathbf{u}_{k+1}] = [\mathbf{f}_c]([\mathbf{u}_k])$$

and let us assume that Equation (3.44) holds for a $k > 0$, i. e.

$$[\mathbf{u}_k] \subset \alpha^k \cdot [\mathbf{x}_0] \tag{3.45}$$

We then want to prove that Equation (3.45) implies

$$[\mathbf{u}_{k+1}] \subset \alpha^{k+1} \cdot [\mathbf{x}_0] \tag{3.46}$$

Then,

$$\begin{aligned}
[\mathbf{u}_{k+1}] &= [\mathbf{f}_c]([\mathbf{u}_k]) \\
&= [\mathbf{J_f}]([\mathbf{u}_k]) \cdot [\mathbf{u}_k] \\
&\subset [\mathbf{J_f}]\left(\alpha^k \cdot [\mathbf{u}_0]\right) \cdot \left(\alpha^k \cdot [\mathbf{u}_0]\right) \\
&\text{since } [\mathbf{J_f}] \text{ is inclusion monotonic and according to Equation (3.45)} \\
&\subset [\mathbf{J_f}]([\mathbf{u}_0]) \cdot \left(\alpha^k \cdot [\mathbf{u}_0]\right) && \text{since } \alpha^k \cdot [\mathbf{x}_0] \subset [\mathbf{x}_0] \\
&= \alpha^k \cdot [\mathbf{J_f}]([\mathbf{x}_0]) \cdot [\mathbf{x}_0] \\
&= \alpha^k \cdot [\mathbf{f}_c]([\mathbf{x}_0]) && \text{according to Theorem 3.3} \\
&\subset \alpha^k \cdot \alpha\,[\mathbf{x}_0] && \text{since } [\mathbf{f}_c]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0] \\
[\mathbf{u}_{k+1}] &\subset \alpha^{k+1} \cdot [\mathbf{x}_0]
\end{aligned}$$

This proves Equation (3.46), and therefore concludes the proof of the convergence property and in turn of Theorem 3.4. ■

Remark 3.5. Despite being possibly less sharp than a Lohner-like algorithm, our iterative centred form algorithm has been designed to be a stability contractor under certain conditions. Since it encloses the states of its associated discrete system, it can prove stability of the latter in a specific neighbourhood $[\mathbf{x}_0]$.

Corollary 3.2. *Consider a function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ *such that* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. *If there exists a box* $[\mathbf{x}_0] \ni \mathbf{0}$, *a real* $\alpha < 1$ *and an integer* $q \geq 1$ *such that* $\left[\mathbf{f}_c^q\right]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$, *then the operator* Ψ *as defined by Equation* (3.47) *is a stability contractor inside* $[\mathbf{x}_0]$.

$$\begin{aligned} \Psi : \mathbb{IR}^n &\to \mathbb{IR}^n \\ [\mathbf{x}] &\mapsto \left[\mathbf{f}_c^q\right]([\mathbf{x}]) \end{aligned} \tag{3.47}$$

Proof. This is a direct consequence of the definition of stability contractor, considering $\mathbf{f}^q$ instead of $\mathbf{f}$. ■

Remark 3.6. Sometimes, a system will require more than a single iteration to converge towards its fixed point, and might temporarily not be contained inside $[\mathbf{x}_0]$, whence the usefulness of this corollary. In other words, the latter allows proving stability of the system $\mathbf{x}_{k+q} = \mathbf{f}^q(\mathbf{x}_k)$. It will be illustrated in Example 3.7.

Proposition 3.6. *Consider a function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ *such that* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$ *and a box* $[\mathbf{x}_0] \ni \mathbf{0}$ *such that the centred form of* $\mathbf{f}$ *is a stability contractor of rate* $\alpha < 1$ *inside* $[\mathbf{x}_0]$. *Then the system described by* $\mathbf{f}$ *is asymptotically stable around the origin.*

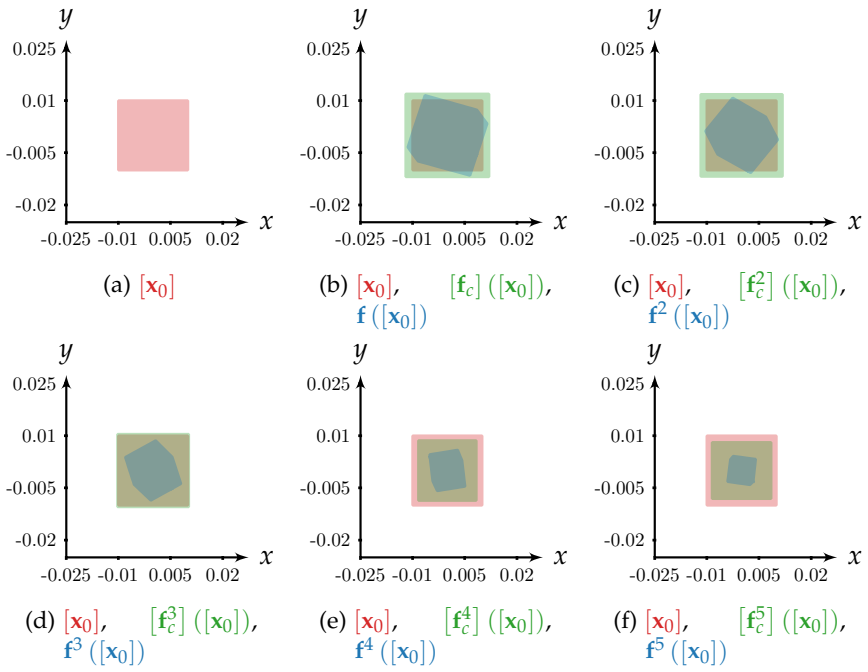
Proof. Since $[\mathbf{f}_c]$ is a stability contractor inside $[\mathbf{x}_0]$, we have $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0]) \subset \alpha \cdot [\mathbf{x}_0]$. Thus, since for all $k \geq 1$, $\mathbf{f}^k(\mathbf{x}_0) \subset \left[\mathbf{f}_c^k\right]([\mathbf{x}_0])$, this proposition is a direct consequence of Corollary 3.1. ■

To illustrate this method, we will only give one example, since discrete systems related to robotics and centred on the origin are not that common. More will come in the next section.

Example 3.7. Consider the system described by Equation (3.48), where $\alpha < 1$ and $\mathbf{R}(x, y, z)$ is a rotation matrix, parametrised by three Tait-Bryan angles following the x-y-z convention. Note that this example does not have any physical meaning.

$$\mathbf{x}_{k+1} = \alpha \cdot \mathbf{f}(\mathbf{x}_k) = \alpha \cdot \mathbf{R}\left(x_{k_0} + a_x, x_{k_1} + a_y, x_{k_2} + a_z\right) \cdot \mathbf{x}_k \tag{3.48}$$

Take a box $[\mathbf{x}_k] \in \mathbb{R}^3$ containing $\mathbf{0}$. Each iteration of the system rotates the latter by an angle depending on the parameters a_x, a_y and a_z and on the size of $[\mathbf{x}_k]$. Additionally, it shrinks the box $[\mathbf{x}_k]$ by a factor α. Therefore, this system is strongly non-linear, but remains centred on the origin, since $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. Intuitively, it is supposed to converge towards $\mathbf{0}$, which can be observed (in blue) thanks to a Monte-Carlo method in Figure 3.12. Note that instead of displaying each simulated point independently, we computed the image set's concave hull, as we did in Section 2.4.4. In green, we drew the box $[\mathbf{x}_k]$, calculated using our algorithm. The red box represents $[\mathbf{x}_0]$ and is displayed at each



(a) $[\mathbf{x}_0]$ (b) $[\mathbf{x}_0]$, $[\mathbf{f}_c]([\mathbf{x}_0])$, $\mathbf{f}([\mathbf{x}_0])$ (c) $[\mathbf{x}_0]$, $[\mathbf{f}_c^2]([\mathbf{x}_0])$, $\mathbf{f}^2([\mathbf{x}_0])$

(d) $[\mathbf{x}_0]$, $[\mathbf{f}_c^3]([\mathbf{x}_0])$, $\mathbf{f}^3([\mathbf{x}_0])$ (e) $[\mathbf{x}_0]$, $[\mathbf{f}_c^4]([\mathbf{x}_0])$, $\mathbf{f}^4([\mathbf{x}_0])$ (f) $[\mathbf{x}_0]$, $[\mathbf{f}_c^5]([\mathbf{x}_0])$, $\mathbf{f}^5([\mathbf{x}_0])$

Figure 3.12: Proving stability of a centred discrete system

step to notice when $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0])$ is contained in $[\mathbf{x}_0]$. We can observe that $\left[\mathbf{f}_c^4\right]([\mathbf{x}_0]) \subset [\mathbf{x}_0]$. According to Proposition 3.6, the system is thus asymptotically stable in the neighbourhood $[\mathbf{x}_0]$.

3.4.2.2 *Discrete iterative centred form*

To prove stability of a larger class of discrete systems, i. e. the ones that are not centred on the origin, we will need to adapt our algorithm according to Remark 3.3.

Theorem 3.5. *Consider a function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ *and a point* $\bar{\mathbf{x}} \in \mathbb{R}^n$ *such that* $\mathbf{f}(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$. *Let* $[\mathbf{x}_0] \ni \bar{\mathbf{x}}$ *be a box of* $\mathbb{R}^n$. *Define the function* $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$ *as follows*

$$\begin{aligned} \mathbf{g} : \mathbb{R}^n &\to \mathbb{R}^n \\ \mathbf{u} &\mapsto \mathbf{f}(\mathbf{u} + \bar{\mathbf{x}}) - \bar{\mathbf{x}} \end{aligned} \tag{3.49}$$

and denote by $\mathbf{J}_\mathbf{g}$ *its Jacobian matrix. The centred form* $\left[\mathbf{f}_c^k\right]([\mathbf{x}_0]) \supset \mathbf{f}^k([\mathbf{x}_0])$ *is given by the following sequence*

$$\begin{aligned} [\mathbf{u}_0] &= [\mathbf{x}_0] - \bar{\mathbf{x}} \\ [\mathbf{A}_0] &= \mathbf{I}_n \\ [\mathbf{u}_{k+1}] &= [\mathbf{g}]([\mathbf{u}_k]) \\ [\mathbf{A}_{k+1}] &= \left[\mathbf{J}_\mathbf{g}\right]([\mathbf{u}_k]) \cdot [\mathbf{A}_k] \\ \left[\mathbf{f}_c^k\right]([\mathbf{x}_0]) &= [\mathbf{A}_k] \cdot [\mathbf{u}_0] + \bar{\mathbf{x}} \end{aligned} \tag{3.50}$$

Proof. Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ and a point $\bar{\mathbf{x}} \in \mathbb{R}^n$ such that $\mathbf{f}(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$. Following Remark 3.3, it follows that for all $k \geq 0$

$$\begin{aligned} & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \\ \Leftrightarrow \quad & \mathbf{x}_{k+1} - \mathbf{f}(\bar{\mathbf{x}}) = \mathbf{f}(\mathbf{x}_k - \bar{\mathbf{x}} + \bar{\mathbf{x}}) - \mathbf{f}(\bar{\mathbf{x}}) \end{aligned} \tag{3.51}$$

Let us define $\mathbf{u}_k = \mathbf{x}_k - \bar{\mathbf{x}}$. It follows from Equation (3.51) that

$$\mathbf{u}_{k+1} = \mathbf{f}(\mathbf{u}_k + \bar{\mathbf{x}}) - \bar{\mathbf{x}} \tag{3.52}$$

and let us define $\mathbf{g}(\mathbf{u}) = \mathbf{f}(\mathbf{u} + \bar{\mathbf{x}}) - \bar{\mathbf{x}}$ which yields

$$\mathbf{u}_{k+1} = \mathbf{g}(\mathbf{u}_k) \tag{3.53}$$

Since $\mathbf{g}(\mathbf{0}) = \mathbf{0}$ and $\mathbf{0} \in [\mathbf{u}_0] = [\mathbf{x}_0] - \bar{\mathbf{x}}$, the centred form $\left[\mathbf{g}_c^k\right]([\mathbf{x}_0]) \supset \mathbf{g}^k([\mathbf{x}_0])$ is given by Theorem 3.3:

$$\left[\mathbf{g}_c^k\right]([\mathbf{x}_0]) = [\mathbf{A}_k] \cdot [\mathbf{u}_0] \tag{3.54}$$

Then, for any $k \geq 0$ we have

$$\begin{aligned} \forall \mathbf{x}_k \in [\mathbf{x}_k], \mathbf{f}(\mathbf{x}_k) &= \mathbf{x}_{k+1} \\ &= \mathbf{u}_{k+1} + \bar{\mathbf{x}} \\ &= \mathbf{g}(\mathbf{u}_k) + \bar{\mathbf{x}} \\ &= \mathbf{g}^{k+1}(\mathbf{u}_0) + \bar{\mathbf{x}} \\ &\in \mathbf{g}^{k+1}([\mathbf{u}_0]) + \bar{\mathbf{x}} \\ \mathbf{f}(\mathbf{x}_k) &\in \left[\mathbf{g}_c^{k+1}\right]([\mathbf{u}_0]) + \bar{\mathbf{x}} \end{aligned} \tag{3.55}$$

Furthermore, for all $k \geq 0$ we have $\mathbf{f}^{k+1}([\mathbf{x}_0]) \subset \mathbf{f}([\mathbf{x}_k])$ (according to Lemma 2.2), thus it follows that

$$\mathbf{f}^{k+1}([\mathbf{x}_0]) \subset \left[\mathbf{g}_c^{k+1}\right]([\mathbf{u}_0]) + \bar{\mathbf{x}} \tag{3.56}$$

Finally, we define the centred form of $\mathbf{f}^k$ as

$$\left[\mathbf{f}_c^k\right]([\mathbf{x}_0]) = [\mathbf{A}_k] \cdot [\mathbf{u}_0] + \bar{\mathbf{x}} \tag{3.57}$$

■

Remark 3.7. From now on, we will assume that the theorems and propositions demonstrated in Section 3.4.2.1 are also valid for a fixed point $\bar{\mathbf{x}} \neq \mathbf{0}$. The demonstrations are very similar to the ones we already gave earlier and do not present any major difficulty. Note that the equilibrium and convergence equations of Definition 3.9 then become

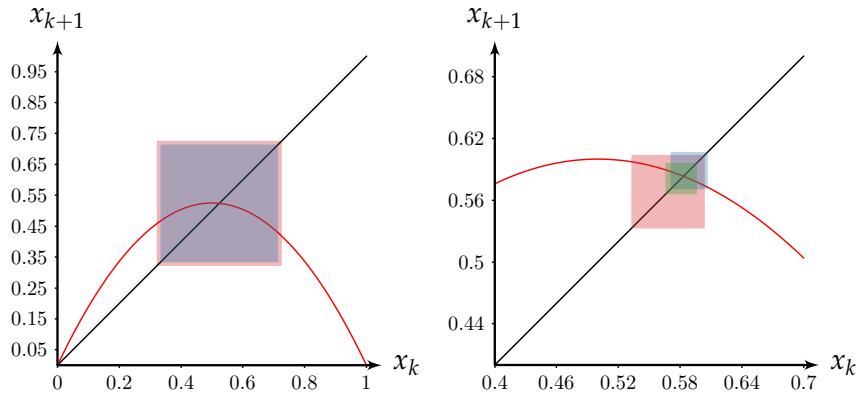
$$\Psi(\bar{\mathbf{x}}) = \bar{\mathbf{x}} \tag{equilibrium}$$

$$\Psi([\mathbf{a}]) \subset \alpha \cdot ([\mathbf{a}] - \bar{\mathbf{x}}) + \bar{\mathbf{x}} \implies \forall k \geq 1, \Psi^k([\mathbf{a}]) \subset \alpha^k \cdot ([\mathbf{a}] - \bar{\mathbf{x}}) + \bar{\mathbf{x}} \tag{convergence}$$

for all $[\mathbf{a}] \ni \bar{\mathbf{x}}$ in a given box $[\mathbf{x}_0] \in \mathbb{IR}^n$.

Example 3.8. Recall the logistic map we introduced in Example 3.4. We could observe that for some values of ρ, the system had a non-zero fixed point. In this example, we will prove stability of the fixed point $\bar{x} = \frac{\rho-1}{\rho}$.

To easily visualise the intervals $[x_0]$ and $\left[f_c^k\right]([x_0])$ in Figures 3.13a and 3.13b, we displayed boxes representing $([x_0], [x_0])$ (in red) and $\left(\left[f_c^k\right]([x_0]), \left[f_c^k\right]([x_0])\right)$ (in blue for $k = 1$ and green for $k = 2$).



(a) Stability of the logistic map for $\rho = 2.1$ and $[x_0] = [-0.2 + \bar{x}, 0.2 + \bar{x}]$

(b) Stability of the logistic map for $\rho = 2.4$ and $[x_0] = [-0.05 + \bar{x}, 0.02 + \bar{x}]$

Figure 3.13: Proving stability of the logistic map

Let us take $\rho = 2.1$ and choose a box $[x_0] = [-0.2 + \bar{x}, 0.2 + \bar{x}]$. Since $[f_c]([x_0]) \subset [x_0]$, the system is asymptotically stable inside $[x_0]$. Note that $[x_0]$ need not be centred on $\bar{x}$ for the algorithm to converge. Take for example $\rho = 2.4$ and $[x_0] = [-0.05 + \bar{x}, 0.02 + \bar{x}]$: the system is asymptotically stable since $\left[f_c^2\right]([x]_0) \subset [x_0]$.

Remark 3.8. Until now, we considered that the fixed point of the system $\bar{\mathbf{x}}$ was perfectly known. However, this is not always the case: even though it can be approached by a fixed point iteration method, there is no guarantee that the result will be exact. Therefore, an interval method must be used to enclose it. We give below an algorithm based on the Newton contractor, presented in [50, Chapter 4]. The latter can be used to find a sharp enclosure for a discrete-time system's fixed point if provided with an initial guess in the form of a box, and the function describing the system.

Denote by $\tilde{\mathbf{x}}$ an approximation of $\bar{\mathbf{x}}$ obtained thanks to a conventional fixed-point iteration method and by $\mathbf{f}$ the function describing the system. Let us define $\epsilon > 0$, which we will use to inflate the enclosure of $\bar{\mathbf{x}}$ in the following algorithm.

Now, assume that we have an enclosure $[\bar{\mathbf{x}}]$ for $\bar{\mathbf{x}}$. Denote by $[\mathbf{x}_0]$ the initial box inside which we aim to prove stability of the system

Algorithm 3 `FixedPointEnclosure (in : $\tilde{\mathbf{x}}$, out : $[\bar{\mathbf{x}}]$)`

$[\tilde{\mathbf{x}}] \leftarrow \tilde{\mathbf{x}} + [-\epsilon, \epsilon]$
$[\bar{\mathbf{x}}] \leftarrow \emptyset$
while $[\bar{\mathbf{x}}] = \emptyset$ **do**
  $[\tilde{\mathbf{x}}] \leftarrow \mathcal{C}^{\infty}_{Newton}([\tilde{\mathbf{x}}])$ — *Contract the initial guess until reaching a fixed point or the empty set*
  $[\tilde{\mathbf{x}}] \leftarrow (1+\epsilon)[\tilde{\mathbf{x}}] - \epsilon[\tilde{\mathbf{x}}]$ — *Inflate the initial guess to hopefully contain* $\bar{\mathbf{x}}$
end while
return $[\bar{\mathbf{x}}]$

described by $\mathbf{f}$. It is then possible to compute $[\mathbf{f}_c]([\mathbf{x}_0])$, using $[\bar{\mathbf{x}}]$ instead of $\bar{\mathbf{x}}$, which we denote by $[\mathbf{f}_c]([\mathbf{x}_0], [\bar{\mathbf{x}}])$. Then, assume that $[\mathbf{f}_c]([\mathbf{x}_0], [\bar{\mathbf{x}}]) \subset [\mathbf{x}_0]$. It follows that

$$\begin{aligned} & \forall \mathbf{y} \in [\bar{\mathbf{x}}], [\mathbf{f}_c]([\mathbf{x}_0], \mathbf{y}) \subset [\mathbf{x}_0] \\ \implies & \exists \mathbf{y} \in [\bar{\mathbf{x}}], \mathbf{y} = \bar{\mathbf{x}}, [\mathbf{f}_c]([\mathbf{x}_0], \bar{\mathbf{x}}) \subset [\mathbf{x}_0] \end{aligned}$$

This implies, in turn, the asymptotic stability of the system around $\bar{\mathbf{x}}$.

Therefore, an enclosure of $\bar{\mathbf{x}}$ suffices to prove stability of a system around that fixed point. However, the neighbourhood inside which the centred form is proven to be a stability contractor might end up being smaller than if $\bar{\mathbf{x}}$ were known.

3.4.2.3 *Application : proving stability of a localisation algorithm*

We will present a robotics use case for our method. When designing the software embedded on a robot, engineers have to make sure that it will actually behave as expected, and ideally find conditions that guarantee its proper functioning. For example, consider a localisation algorithm: its role is to provide the other programs running on the robot with an estimation $\hat{\mathbf{x}}$ of the latter's state $\mathbf{x}$. It goes without saying that the more precise and accurate the estimation, the better the robot's behaviour. Now, a state estimator processes discrete sensory inputs, to estimate a continuous state. Ideally, the latter should be a point attractor, i. e. a stable fixed point, for the estimation. Since we focus on discrete systems in this section, we will consider that the robot stays still in this example.

This is true for most sensors used in robotics: IMU, GPS, pressure sensor, sonar...

Therefore, consider a robot which position is denoted by $\mathbf{x} = (x, y)$, embedding a state estimator which estimation is denoted by $\hat{\mathbf{x}}_k$, k being a time index, and $\hat{\mathbf{x}}_0$ the original state estimation. The robot performs range measurements with two beacons placed at points

$\mathbf{a} = (a_x, a_y)$ and $\mathbf{b} = (b_x, b_y)$ to localise itself (see Figure 3.14). Let $\mathbf{y} = (y_a, y_b)$ be the range measurements, defined by

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \begin{pmatrix} (x - a_x)^2 + (y - a_y)^2 \\ (x - b_x)^2 + (y - b_y)^2 \end{pmatrix} \tag{3.58}$$

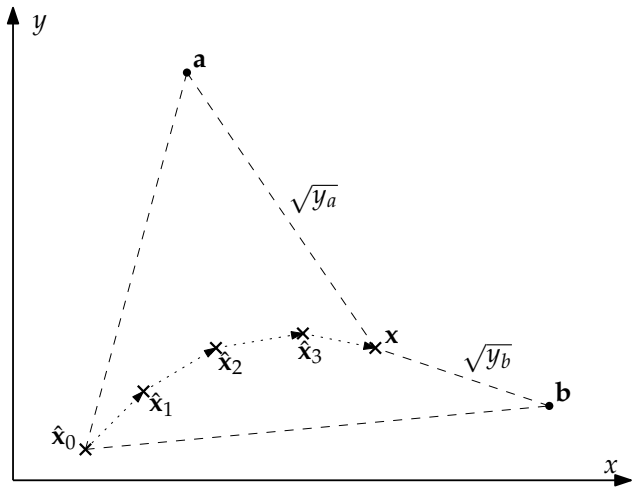


Figure 3.14: Range-only localisation – stability of the state estimation algorithm

3.4.2.3.1 LOCALISATION BASED ON NEWTON'S METHOD

To begin with, let us consider a state estimator based on the Newton-Raphson method. The state estimation is improved iteratively by successive measurements defined by the following equation

$$\hat{\mathbf{x}}_{k+1} = \mathbf{f}(\hat{\mathbf{x}}_k) = \hat{\mathbf{x}}_k + \mathbf{J}_{\mathbf{h}}^{-1}(\hat{\mathbf{x}}_k) \cdot (\mathbf{h}(\mathbf{x}) - \mathbf{h}(\hat{\mathbf{x}}_k)) \tag{3.59}$$

where $\mathbf{J}_{\mathbf{h}}$ denotes the Jacobian matrix of $\mathbf{h}$.

The goal is to prove that this localisation method will make the state estimation $\hat{\mathbf{x}}_k$ converge towards the actual state of the robot $\mathbf{x}$.

It is straightforward that $\mathbf{x}$ is a fixed point of $\mathbf{f}$. Therefore, by choosing a box $[\hat{\mathbf{x}}_0] \ni \mathbf{x}$, our method should be able to prove stability of the system. Take for example $\mathbf{a} = (-5, 10)$, $\mathbf{b} = (5, 10)$, and $[\hat{\mathbf{x}}_0] = ([2.75, 3.25], [3.75, 4.25])$. Using Theorem 3.5, we get after one iteration $[\mathbf{f}_c]([\hat{\mathbf{x}}_0]) = ([2.8, 3.2], [3.8, 4.15]) \subset [\hat{\mathbf{x}}_0]$, which proves stability of this localisation method inside $[\hat{\mathbf{x}}_0]$.

Now, this information is not extremely useful *per se*. However, by scanning the whole state space, one could find the set of all stable initial conditions, i. e. the basin of attraction of the system for a given robot's position $\mathbf{x}$, inside which the localisation method will converge

towards the actual robot's position. Similarly, by taking uncertainties in the robot's position $[\mathbf{x}]$ into account, one can determine the system's stability region, i. e. the region of the parameter space (here the space of the robot's actual position) where the method is stable. This example will be continued in Section 3.5.2.

3.4.2.3.2 DISCRETE EXTENDED KALMAN FILTER : THEORY

Now, in industry, one of the most used methods to perform state estimation is the Kalman filter, and its non-linear form called Extended Kalman Filter (EKF). The goal of this example is to illustrate how our method can be used to prove stability of an EKF, which will require more work than the previous method. We assume that the reader is familiar with the basics of Kalman filtering.

Otherwise, we refer the reader to [48], [121]

In addition to the robot's state estimation, the EKF estimates the accuracy of the latter, in the form of a covariance matrix $\mathbf{\Gamma}_k$. The latter depends on the variance of each state variable, and their covariances. Therefore, the actual state of the discrete EKF is composed of the robot's state estimation and of the estimated variances and covariances forming $\mathbf{\Gamma}_k$:

$$\hat{\mathbf{x}}_k = \left(\hat{x}_k, \hat{y}_k, \sigma_{x_k}, \sigma_{y_k}, \sigma_{xy_k}\right) \tag{3.60}$$

Usually, a Kalman filter performs two main steps: the prediction step, during which the state estimation evolves according to a dynamical model of the robot; and a correction step, during which sensory measurements are used to correct the prediction. Both the prediction and correction steps are parametrised by a covariance matrix $\mathbf{\Gamma}_\alpha$ and $\mathbf{\Gamma}_\beta$ to take into account the process (or model) and measurement noise (or uncertainties). Figure 3.15 sums up the notations introduced so far.

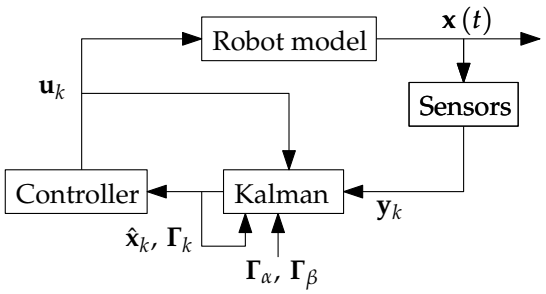


Figure 3.15: Block diagram of a Kalman filter based localisation & control architecture

We assumed that the robot was not moving; therefore, solely the correction step should be executed in our case. However, this would make the covariance matrix converge towards a matrix close to the

null matrix, which implies that the state's estimation is almost 100% certain, which is not the case. Therefore, we will consider that a process noise impacts the model. The covariance matrix should then converge towards a better-conditioned matrix. This choice is perfectly justified and accepted from an engineering perspective: this way of tuning the Kalman filter's parameters is referred to as "adding a Brownian motion".

This choice is also strategic: by shifting the fixed point of the covariance matrix away from the null matrix, we can therefore propose an initial enclosure for the state $\hat{\mathbf{x}}_0$ that does not contain $\mathbf{0}$ for the terms σ_{x_k}, σ_{y_k} and σ_{xy_k}. Again, these terms being null would imply that the state is 100% certain, and the box of initial position would therefore not be contracted by our stability contractor.

Let us now propose our EKF for the localisation problem presented earlier

$$\begin{aligned}
\begin{pmatrix} \hat{x}_{k+1} \\ \hat{y}_{k+1} \end{pmatrix} &= \begin{pmatrix} \hat{x}_k \\ \hat{y}_k \end{pmatrix} + \mathbf{K}_k \cdot \tilde{\mathbf{z}}_k && \text{(corrected position)} \\
\mathbf{\Gamma}_{k+1} &= (\mathbf{I} - \mathbf{K}_k \cdot \mathbf{H}_k) \cdot \mathbf{G}_k && \text{(corrected covariance)} \\
\tilde{\mathbf{z}}_k &= \mathbf{y}_k - \mathbf{h}\left(\hat{\mathbf{x}}_k\right) && \text{(innovation)} \\
\mathbf{G}_k &= \mathbf{\Gamma}_k + \mathbf{\Gamma}_\alpha && \text{(predicted covariance)} \\
\mathbf{K}_k &= \mathbf{G}_k \cdot \mathbf{H}_k^{\mathrm{T}} \cdot \mathbf{S}_k^{-1} && \text{(Kalman gain)} \\
\mathbf{S}_k &= \mathbf{H}_k \cdot \mathbf{G}_k \cdot \mathbf{H}_k^{\mathrm{T}} + \mathbf{\Gamma}_\beta && \text{(innovation's covariance)} \\
\mathbf{H}_k &= \frac{\partial \mathbf{h}}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}_k\right) && \text{(observation matrix)}
\end{aligned}$$

We recall below the form of $\mathbf{\Gamma}_k$

$$\mathbf{\Gamma}_k = \begin{bmatrix} \sigma_{x_k}^2 & \sigma_{xy_k} \\ \sigma_{xy_k} & \sigma_{y_k}^2 \end{bmatrix} \tag{3.61}$$

Now, the equations given above can be transformed into a discrete equation of the form $\hat{\mathbf{x}}_{k+1} = \mathbf{f}\left(\hat{\mathbf{x}}_k\right)$, although we will not detail this cumbersome process here and we highly recommand to execute it with a computer program. Once this is achieved, it only remains for us to choose an initial box $\hat{\mathbf{x}}_0$, which must contain the discrete system's fixed point. The latter can be approximated using Remark 3.8.

3.4.2.3.3 DISCRETE EXTENDED KALMAN FILTER : NUMERICAL APPLICATION

As a numerical application, take the following parameters:

$$\mathbf{x} = \mathbf{0} \tag{3.62}$$

$$[\hat{\mathbf{x}}_0] = \begin{pmatrix} [-\varepsilon_1, \varepsilon_1] \\ [-\varepsilon_1, \varepsilon_1] \\ [\sigma_0 - \varepsilon_2, \sigma_0 + \varepsilon_2] \\ [\sigma_0 - \varepsilon_2, \sigma_0 + \varepsilon_2] \\ [-\varepsilon_2, \varepsilon_2] \end{pmatrix} \tag{3.63}$$

$$\varepsilon_1 = 1 \times 10^{-3} \tag{3.64}$$

$$\varepsilon_2 = 5 \times 10^{-5} \tag{3.65}$$

$$\sigma_0 = 3.66 \times 10^{-3} \tag{3.66}$$

$$\mathbf{\Gamma}_\alpha = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{3.67}$$

$$\mathbf{\Gamma}_\beta = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.68}$$

$$\mathbf{a} = (-5, 5) \tag{3.69}$$

$$\mathbf{b} = (5, 5) \tag{3.70}$$

Although the initial box is very small, which is necessary to avoid wrapping effect, we prove that the EKF converges towards the system's actual fixed point. So does the estimated position towards the actual one.

$$[\mathbf{f}_c]([\hat{\mathbf{x}}_0]) = \begin{pmatrix} \left[-3.5 \times 10^{-4}, 3.6 \times 10^{-4}\right] \\ \left[-3.5 \times 10^{-4}, 3.6 \times 10^{-4}\right] \\ \left[\sigma_0 - 4.63 \times 10^{-5}, \sigma_0 + 4.68 \times 10^{-5}\right] \\ \left[\sigma_0 - 4.73 \times 10^{-5}, \sigma_0 + 4.79 \times 10^{-5}\right] \\ \left[-1.48 \times 10^{-5}, 1.49 \times 10^{-5}\right] \end{pmatrix} \subset [\hat{\mathbf{x}}_0]$$

Again, this result is not very useful, but can be used to approximate the system's stability region or basin of attraction. This will be done for the Newton method based estimator in Section 3.5.2.

3.4.3 *Proving stability of continuous-time systems*

We will now address continuous systems, and show how the method presented earlier can also be used to analyse their stability.

3.4.3.1 *Discretising the continuous-time system*

Since the discrete centred form allows proving stability for a discrete-time system, we will show that it can also be used to prove stability of a continuous-time system. In particular, we will show that the stability of a discretised system implies that of its continuous-time counterpart.

Definition 3.10 (Discretised system)**.** Consider a continuous-time system described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$, and denote by ϕ its flow function. Define a time step $\delta > 0$ and a sequence $(t_k)_{k\geq 0}$ such that $t_k = k\delta$. We then define the *discretised system* associated to $\mathbf{f}$ as follows

$$\mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k) \tag{3.71}$$

where for all $k \geq 0$, $\mathbf{x}_k = \mathbf{x}(t_k)$ and $\hat{\mathbf{f}}$ is defined by

$$\begin{aligned} \hat{\mathbf{f}} : \mathbb{R}^n &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto \phi(\delta, \mathbf{x}) \end{aligned} \tag{3.72}$$

Theorem 3.6. *Consider a continuous-time system described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$. Denote by $\hat{\mathbf{f}}$ its associated discretised system, and by δ its time step. Assume that $\bar{\mathbf{x}}$ is an equilibrium point of $\mathbf{f}$, and choose a box $[\mathbf{x}_0] = [\mathbf{x}](0)$ containing $\bar{\mathbf{x}}$. If the centred form $\left[\hat{\mathbf{f}}_c\right]$ is a stability contractor of rate $\alpha < 1$ inside $[\mathbf{x}_0]$, then the continuous system described by $\mathbf{f}$ is asymptotically stable in the neighbourhood $[\mathbf{x}_0]$*

This theorem states that the stability of the discretised system implies that of its continuous counterpart. Before proceeding with the proof of this theorem, let us introduce the following lemmas and theorem.

Lemma 3.1. *Consider a continuous-time system described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, where $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$, and denote by $\mathbf{J}_\mathbf{f}$ its Jacobian matrix. Define the associated discretised system of time step δ as $\mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k)$. Then the space derivative of $\hat{\mathbf{f}}$ at each discrete state $\mathbf{x}_k$ corresponds to the solution of the variational equation of the system integrated from t_k to t_{k+1}, choosing $\mathbf{U}(t_k) = \mathbf{I}_n$*

$$\dot{\mathbf{U}}(t) = \mathbf{J}_\mathbf{f}(\mathbf{x}(t)) \cdot \mathbf{U}(t) \tag{3.73}$$

$$\mathbf{J}_{\hat{\mathbf{f}}}(\mathbf{x}_k) = \mathbf{U}(t_{k+1}) \tag{3.74}$$

Proof. Denote by ϕ the flow function of the system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. From Theorem 2.2, it follows that

$$\frac{\partial \phi}{\partial \mathbf{x}}(t_{k+1}, \mathbf{x}_0) \cdot \mathbf{U}_0 = \mathbf{U}(t_{k+1})$$

Moreover, according to Definition 3.10, we have

$$\begin{aligned} \hat{\mathbf{f}}(\mathbf{x}_k) &= \phi(\delta, \mathbf{x}_k) \\ &= \phi(t_k + \delta, \mathbf{x}_0) \\ &= \phi(t_{k+1}, \mathbf{x}_0) \end{aligned}$$

Differentiating this with respect to $\mathbf{x}$ and choosing $\mathbf{U}_0 = \mathbf{I}_n$ yields

$$\begin{aligned} \hat{\mathbf{f}}(\mathbf{x}_k) &= \frac{\partial \phi}{\partial \mathbf{x}}(t_{k+1}, \mathbf{x}_0) \\ &= \mathbf{U}(t_{k+1}) \end{aligned}$$

■

Lemma 3.2. *Consider a continuous-time system described by* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$*, where* $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^n)$*, and assume that* $\bar{\mathbf{x}}$ *is an equilibrium state of* $\mathbf{f}$*. Then it is also a fixed point of the discretised system* $\mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k)$*.*

Proof. Assume that $\bar{\mathbf{x}}$ is an equilibrium state of the system, then any trajectory initialised at $\bar{\mathbf{x}}$ remains on $\bar{\mathbf{x}}$. It follows that $\forall t \geq 0$, $\mathbf{x}(t) = \bar{\mathbf{x}}$, which is thus true for all t_k, $k \geq 0$. ■

Definition 3.11, Lemma 3.3 and Theorem 3.7 are presented and derived in [41, Chapter 17].

Definition 3.11 (Lipschitz)**.** Let $\mathcal{O}$ be an open set of $\mathbb{R}^n$. A function $\mathbf{f} : \mathcal{O} \to \mathbb{R}^n$ is *Lipschitz* on $\mathcal{O}$ if there exists a constant $K > 0$ such that

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{O},\ \|\mathbf{f}(\mathbf{y}) - \mathbf{f}(\mathbf{x})\| \leq K\|\mathbf{y} - \mathbf{x}\| \tag{3.75}$$

Lemma 3.3. *Let the function* $\mathbf{f} : \mathcal{O} \to \mathbb{R}^n$ *be* $\mathcal{C}^1$*. Then* $\mathbf{f}$ *is locally Lipschitz, i.e. each point* $\mathbf{x} \in \mathcal{O}$ *has a neighbourhood* $\mathcal{O}'$ *such that* $\mathbf{f} : \mathcal{O}' \to \mathbb{R}^n$ *is Lipschitz.*

Theorem 3.7. *Consider a Lipschitz function with a constant* K *over an open* $\mathcal{O} \subset \mathbb{R}^n$*. Let* $\mathbf{x}_a(t)$ *and* $\mathbf{x}_b(t)$ *be solutions of* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *remaining in* $\mathcal{O}$ *and defined on the time interval* $[t_0, t_f]$*. Then, for all* $t \in [t_0, t_f]$ *we have*

$$\|\mathbf{x}_a(t) - \mathbf{x}_b(t)\| \leq \|\mathbf{x}_a(t_0) - \mathbf{x}_b(t_0)\| e^{K(t-t_0)} \tag{3.76}$$

Remark 3.9. This theorem states that two trajectories initialised close together cannot move away from one another faster than exponentially.

To find a value for K, define $\mathbf{u}(t) = \mathbf{x}_a(t) - \mathbf{x}_b(t)$ for all $t \in [t_0, t_f]$. Then we have $\dot{\mathbf{u}}(t) = \mathbf{f}(\mathbf{x}_a(t)) - \mathbf{f}(\mathbf{x}_b(t))$. Now, let us define v as the maximum velocity of the system along $\mathbf{x}_a(t)$ and $\mathbf{x}_b(t)$:

$$v = \max_{\mathbf{x} \in \{\mathbf{x}_a, \mathbf{x}_b\}} \left(\max_{t \in [t_0, t_f]} \left(\|\mathbf{f}(\mathbf{x}(t))\| \right) \right)$$

Then, we have

$$\begin{aligned} \|\dot{\mathbf{u}}(t)\| &= \|\dot{\mathbf{x}}_a(t) - \dot{\mathbf{x}}_b(t)\| \\ &= \|\mathbf{f}(\mathbf{x}_a(t)) - \mathbf{f}(\mathbf{x}_b(t))\| \\ &\leq \|\mathbf{f}(\mathbf{x}_a(t))\| + \|\mathbf{f}(\mathbf{x}_b(t))\| \\ &\leq 2v \end{aligned}$$

It follows by integrating the last step that

$$\begin{aligned} &\|\mathbf{u}(t)\| \leq \|\mathbf{u}(t_0)\| e^{2v(t-t_0)} \\ \Leftrightarrow\ &\|\mathbf{x}_a(t) - \mathbf{x}_b(t)\| \leq \|\mathbf{x}_a(t_0) - \mathbf{x}_b(t_0)\| e^{2v(t-t_0)} \end{aligned}$$

This result means that two solutions cannot move away from one another faster than two times the system's maximum velocity along these solutions.

Proof of Theorem 3.6. Consider a continuous-time system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, such that $\mathbf{f}$ is $\mathcal{C}^1$ inside $\mathcal{O} \subset \mathbb{R}^n$. According to Definition 3.10, we can define its discretised counterpart by choosing a time step δ, which yields $\mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k)$. Lemma 3.1 defines its Jacobian matrix $\mathbf{J}_{\hat{\mathbf{f}}}(\mathbf{x}_k)$ at each time step.

Now, let us consider that $\bar{\mathbf{x}}$ is an equilibrium state for the continuous system, and therefore for its associated discrete system (according to Lemma 3.2). Then by applying Theorem 3.4, we are able to prove stability of the latter, in the sense that if we are able to find $[\mathbf{x}_0] \ni \bar{\mathbf{x}}$ such that $\left[\hat{\mathbf{f}}_c\right]([\mathbf{x}_0]) \subset [\mathbf{x}_0]$, this prove stability of the discrete system in the neighbourhood $[\mathbf{x}_0]$. In other words, the sequence $\mathcal{S} = ([\mathbf{x}_k])_{k\geq 0}$ converges towards $\bar{\mathbf{x}}$.

Therefore, we have proved stability of the discretised system, not the continuous one yet. To do so, we need to prove that the trajectory cannot diverge in between the time step. Now, choose a box $[\mathbf{x}_k]$ in $\mathcal{S}$. According to Lemma 3.3, $\mathbf{f}$ is Lipschitz, with a constant $K > 0$, inside $\mathcal{O} \subset \mathbb{R}^n$. We know that $\bar{\mathbf{x}} \in [\mathbf{x}_k]$ is a solution of the continuous system remaining in $\mathcal{O}$. Take any point $\mathbf{y}_0$ inside $[\mathbf{x}_k]$, and assume that its trajectory denoted by $\mathbf{y}(t)$ remains in $\mathcal{O}$ over the time interval $[t_k, t_{k+1}]$ (if this is not the case, one just needs to reduce the time step). Then, according to Theorem 3.7, it follows that

$$\begin{aligned} \forall t \in [t_k, t_{k+1}],\ \|\mathbf{y}(t) - \bar{\mathbf{x}}\| &\leq \|\mathbf{y}_0 - \bar{\mathbf{x}}\| e^{K(t-t_k)} \\ &\leq \|\mathbf{y}_0 - \bar{\mathbf{x}}\| e^{K\delta} \end{aligned} \tag{3.77}$$

Note that $e^{K\delta}$ is a constant valid for all k.

Now, let us choose $\mathbf{y}_0$ close enough to $\bar{\mathbf{x}}$, e. g. $\|\mathbf{y}_0 - \bar{\mathbf{x}}\| = \|[\mathbf{x}_k] - \bar{\mathbf{x}}\| e^{-K\delta}$. Then, there exists a time $t_N > t_k$ and a box $[\mathbf{x}_N]$ such that $\mathbf{y}_0 \in [\mathbf{x}_N]$. Furthermore, Equation (3.77) becomes

$$\begin{aligned} \forall t \in [t_k, t_{k+1}],\ \|\mathbf{y}(t) - \bar{\mathbf{x}}\| &\leq \|\mathbf{y}_0 - \bar{\mathbf{x}}\| e^{K\delta} \\ &\leq \|[\mathbf{x}_k] - \bar{\mathbf{x}}\| \end{aligned} \tag{3.78}$$

In other words, for all $[\mathbf{x}_k]$ of S, there exists a time step indexed by $N > k$ such that all trajectories initialised in $[\mathbf{x}_N]$ will remain inside $[\mathbf{x}_k]$. This proves that past the time $t_N = N\delta$, the system's trajectory cannot leave the box $[\mathbf{x}_k]$ (see Figure 3.16).

Since the sequence $\mathcal{S}$ tends towards $\bar{\mathbf{x}}$, we have $\lim_{k\to\infty} [\mathbf{x}_k] = \bar{\mathbf{x}}$ and thus $\lim_{t\to\infty} \mathbf{y}(t) = \bar{\mathbf{x}}$, which proves asymptotic stability of the continuous system. ■

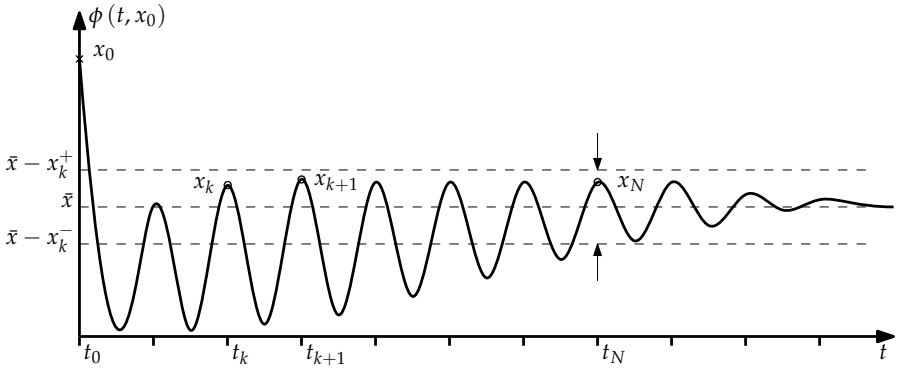


Figure 3.16: Illustration of the proof of Theorem 3.6

3.4.3.2 *Application : proving stability of a simple pendulum*

Let us take the simple pendulum as an example of a continuous system, described by Equation (2.41), with the damping coefficient $d = 0.5$. In the next paragraphs, we will detail the method for proving stability of a continuous-time system.

3.4.3.2.1 BUILDING THE DISCRETISED SYSTEM

As stated by Theorem 3.6, proving stability of a continuous system comes down to proving stability of its discretised counterpart, described by Equation (3.79), where δ is the discretisation step.

$$\begin{aligned} \hat{\mathbf{f}} : \mathbb{R}^2 &\to \mathbb{R}^2 \\ \mathbf{x} &\mapsto \phi(\delta, \mathbf{x}) \end{aligned} \tag{3.79}$$

It is straightforward that $\mathbf{0}$ is an equilibrium point for both the continuous-time and the discretised system. Therefore, to prove stability of the latter, according to Theorems 3.3 and 3.4, we need to compute $\mathbf{J}_{\hat{\mathbf{f}}}$. To do so, we will use Lemma 3.1, and use the discretised variational equation of the system.

Thus, to be able to compute $\left[\hat{\mathbf{f}}_c\right]([\mathbf{x}_0])$, $[\mathbf{x}_0]$ being an initial box, we will need to iteratively integrate the following pair of equations for a time δ

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \\ \dot{\mathbf{U}}(t) = \mathbf{J}_{\mathbf{f}}(\mathbf{x}(t)) \cdot \mathbf{U}(t) \end{cases} \tag{3.80}$$

starting from the initial conditions $[\mathbf{x}](0) = [\mathbf{x}_0]$ and $\mathbf{U}(0) = \mathbf{I}_2$. Note that this pair of equations is composed of the system's model and its associated variational equation (see Paragraph 2.2.2.3.2).

3.4.3.2.2 IMPLEMENTING THE PROBLEM IN CAPD

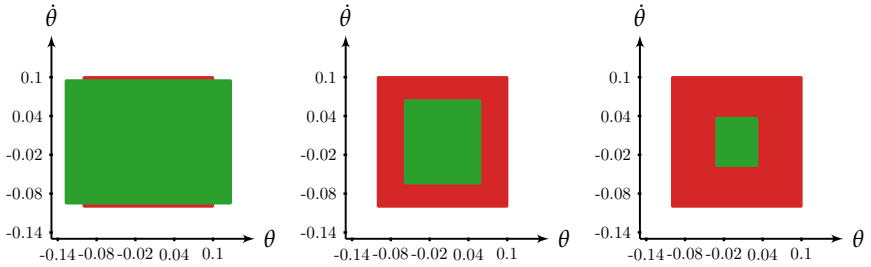
The following piece of code implements Equation (3.80) in CAPD. We chose $[\mathbf{x}_0] = ([-0.1, 0.1], [-0.1, 0.1])$ as an initial condition, and $\delta = 1\,\mathrm{s}$ as an integration step.

```
1   void f(Node, Node x[], int, Node f[], int, Node params[], int
    ) {
2     f[0] = x[1];
3     f[1] = -sin(x[0]) - 0.5 * x[1];
4   }
5
6   int main(int argc, char** argv) {
7     int dimX = 2, dimF = 2, dimP = 0;
8     double delta = 1; // δ = 1 s
9     capd::IMap vf(f, dimX, dimF, dimP);
10    capd::IOdeSolver solver(vf, 30);
11    capd::ITimeMap timeMap(solver);
12
13    capd::IVector x0 = {{-0.1, 0.1}, {-0.1, 0.1}}; // Initial
    state of the pendulum
14    capd::IMatrix A = {{1, 0}, {0, 1}};
15    C1Rect2Set set(x0, 0); // we chose a doubleton set to
    represent the solutions. Note the C1..., which allows
    computing the first derivative of the solution's trajectory
16
17    for (int k = 0; k < 3; ++k) { // here, 3 iterations are
    enough to get stability
18      timeMap(set.getCurrentTime() + delta, set); // integrate
    Equation (3.80) for a duration δ
19      A = ((capd::IMatrix) set) * A; // extract the solution of
     the variational equation [U_{k+1}] and compute
    [A_{k+1}] = [J_f̂] ([x_k]) · [A_k]
20      capd::IVector fck = A * x0; // compute [f̂_c] ([x_0])
21      if (x0[0].contains(fck[0]) and x0[1].contains(fck[1])) {
22        std::cout << "The system is stable inside [x0]" << std
    ::endl;
23        break;
24      }
25    }
26    return EXIT_SUCCESS;
27  }
28
```

Program 3.2: Implementing a continuous system and its variational equation

3.4.3.2.3 RESULTS

In Figure 3.17, we represented the three first steps of the algorithm given above. One can notice that $\left[\hat{\mathbf{f}}_c^2\right]([\mathbf{x}_0]) \subset [\mathbf{x}_0]$, which proves stability of the damped pendulum inside $[\mathbf{x}_0]$.



(a) $[\mathbf{x}_0]$, $\left[\hat{\mathbf{f}}_c\right]([\mathbf{x}_0])$ (b) $[\mathbf{x}_0]$, $\left[\hat{\mathbf{f}}_c^2\right]([\mathbf{x}_0])$ (c) $[\mathbf{x}_0]$, $\left[\hat{\mathbf{f}}_c^3\right]([\mathbf{x}_0])$

Figure 3.17: Results of the continuous-time pendulum integration proving its stability

3.4.4 *Proving stability of uncertain hybrid limit cycles*

In Section 3.4.2.2, we provided a method for proving stability of a discrete system inside a neighbourhood $[\mathbf{x}_0]$. Therefore, according to Section 3.2.3, we can also prove stability of a hybrid limit cycle. Indeed, if we can evaluate the Poincaré map of a system and its derivative in a guaranteed way, we can apply the method described in Section 3.2.3.

In the following paragraphs, we will introduce a periodic hybrid system, and we will prove stability of its limit cycle. We will make use of CAPD to evaluate the various Poincaré maps involved in the computations and their derivatives, and we will apply the method presented in Section 3.2.3.

3.4.4.1 *Presentation of the system*

Consider an AUV cruising in a lake. Before launching the robot, the user implemented virtual fences in the robot's mission planning software to prevent the latter from grounding on the shores (see Figure 3.18). This mission planning program takes the form of a Finite State Machine (FSM) which can be summed up as follows: the robot heads East for a certain amount of time denoted by T, before turning north until reaching the northern virtual fence. Then, it heads South-West until reaching another fence, before heading East again and starting over this cycle. Our goal is to prove that the latter is stable.

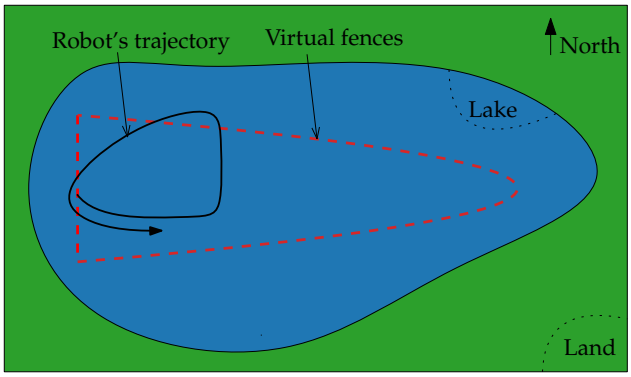


Figure 3.18: Schematic representation of the robot evolving inside the virtual fences

3.4.4.2 *Formalisation of the problem*

The robot's continuous state $\mathbf{x} \in \mathbb{R}^4$ is composed of its position given by x and y, its orientation θ, and a time variable τ as usual. The robot is controlled by a FSM, represented in Figure 3.19, which state q_i, $i \in \{0, 1, 2\}$ corresponds to the discrete state of the system: in q_0, the robot heads East, in q_1 North and in q_2 South-West. Let us define the three transitions $e_0 = (q_0, q_1)$, $e_1 = (q_1, q_2)$ and $e_2 = (q_2, q_0)$. The three guards of the system are defined by Equations (3.81) to (3.83): G_{e_0} corresponds to the limit time $T = 5$ after which the robots switches from East to North as a heading, G_{e_1} corresponds to the northern virtual fence equation, and G_{e_2} to the western one. The vector fields are given by Equation (3.84) and the reset functions by Equations (3.85) to (3.87).

The constants used in Equations (3.81) to (3.83) are arbitrary, as well as the headings used in the FSM. Some combinations of constants might not lead to a periodic trajectory though.

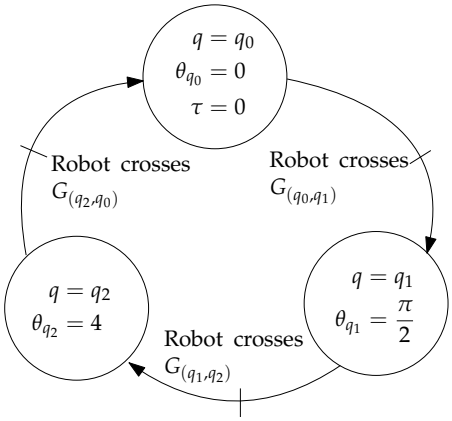


Figure 3.19: FSM controlling the robot

$$G_{e_0} = \left\{ \mathbf{x} \in \mathbb{R}^4 \,\middle|\, \tau - 5 = 0 \right\} \tag{3.81}$$

$$G_{e_1} = \left\{ \mathbf{x} \in \mathbb{R}^4 \,\middle|\, x - y^2 + 20 = 0 \right\} \tag{3.82}$$

$$G_{e_2} = \left\{ \mathbf{x} \in \mathbb{R}^4 \,\middle|\, x + 2 = 0 \right\} \tag{3.83}$$

$$\dot{\mathbf{x}} = \mathbf{f}_{q_i}(\mathbf{x}) = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ \sin(\theta_{q_i} - \theta) \\ 1 \end{pmatrix} \tag{3.84}$$

$$R_{e_0}(\mathbf{x}) = (x, y, \theta, 0) \tag{3.85}$$
$$R_{e_1}(\mathbf{x}) = \mathbf{x} \tag{3.86}$$
$$R_{e_2}(\mathbf{x}) = \mathbf{x} \tag{3.87}$$

3.4.4.3 *Discretisation using Poincaré maps*

According to the method presented in Section 3.2.3, we shall now define the Poincaré maps of the system, associated to the guards G_{e_i} (see Equations (3.88) to (3.90)).

$$\begin{aligned} \Pi_{q_0} : \mathbb{R}^4 &\to G_{e_0} \\ \mathbf{x} &\mapsto \phi_{q_0}\left(\tau_{G_{e_0}}(\mathbf{x}), \mathbf{x}\right) \end{aligned} \tag{3.88}$$

$$\begin{aligned} \Pi_{q_1} : \mathbb{R}^4 &\to G_{e_1} \\ \mathbf{x} &\mapsto \phi_{q_1}\left(\tau_{G_{e_1}}(\mathbf{x}), \mathbf{x}\right) \end{aligned} \tag{3.89}$$

$$\begin{aligned} \Pi_{q_2} : \mathbb{R}^4 &\to G_{e_2} \\ \mathbf{x} &\mapsto \phi_{q_2}\left(\tau_{G_{e_2}}(\mathbf{x}), \mathbf{x}\right) \end{aligned} \tag{3.90}$$

Note that Π_{q_0} simply comes down to integrating ϕ_{q_0} during 5 time units (say, minutes).

Let us now define the hybrid Poincaré map Π:

$$\begin{aligned} \Pi : \mathbb{R}^4 &\to G_{e_2} \\ \mathbf{x} &\mapsto \Pi_{q_2} \circ \Pi_{q_1} \circ \Pi_{q_0}(\mathbf{x}) \end{aligned} \tag{3.91}$$

and the associated discrete system

$$\mathbf{x}_{k+1} = \Pi(\mathbf{x}_k) \tag{3.92}$$

where $\mathbf{x}_0 \in \mathbb{R}^4$ is the initial state of the system.

3.4.4.4 *Proving stability of the hybrid limit cycle*

To compute Poincaré maps and their derivatives, we used the CAPD library. In particular, given an initial enclosure $[\mathbf{x}_0]$ containing a fixed

point $[\bar{\mathbf{x}}]$ of the system (possibly estimated using the procedure described in Remark 3.8), we can compute an enclosure for

$$\left[\Pi_{q_0}\right]\left([\mathbf{x}_0]\right) = [\mathbf{x}_a] \tag{3.93}$$
$$\left[\Pi_{q_1}\right]\left([\mathbf{x}_a]\right) = [\mathbf{x}_b] \tag{3.94}$$
$$\left[\Pi_{q_2}\right]\left([\mathbf{x}_b]\right) = [\mathbf{x}_1] \tag{3.95}$$

Therefore, we get

$$[\Pi]\left([\mathbf{x}_0]\right) = [\mathbf{x}_1] \tag{3.96}$$

Similarly, let us define the derivatives of the local Poincaré maps:

$$\left[\frac{\partial \Pi_{q_0}}{\partial \mathbf{x}}\right]\left([\mathbf{x}_0]\right) = [\mathbf{M}_a] \tag{3.97}$$
$$\left[\frac{\partial \Pi_{q_1}}{\partial \mathbf{x}}\right]\left([\mathbf{x}_a]\right) = [\mathbf{M}_b] \tag{3.98}$$
$$\left[\frac{\partial \Pi_{q_2}}{\partial \mathbf{x}}\right]\left([\mathbf{x}_b]\right) = [\mathbf{M}_c] \tag{3.99}$$

which yields, according to the chain rule

$$\left[\frac{\partial \Pi}{\partial \mathbf{x}}\right]\left([\mathbf{x}_0]\right) = [\mathbf{M}_c] \cdot [\mathbf{M}_b] \cdot [\mathbf{M}_a] \tag{3.100}$$

To apply Theorem 3.5, we simply need to compute $[\Pi]\left([\bar{\mathbf{x}}]\right)$, and then we get the centred form of Π

$$[\Pi_c]\left([\mathbf{x}_0]\right) = \left[\frac{\partial \Pi}{\partial \mathbf{x}}\right]\left([\mathbf{x}_0]\right) \cdot \left([\mathbf{x}] - [\bar{\mathbf{x}}]\right) + [\Pi]\left([\bar{\mathbf{x}}]\right) \tag{3.101}$$

Finally, if we have $\left[\frac{\partial \Pi}{\partial \mathbf{x}}\right]\left([\mathbf{x}_0]\right) \subset [\mathbf{x}_0]$, then the Poincaré map is stable inside $[\mathbf{x}_0]$, which implies stability of the hybrid limit cycle when initialised inside the latter. In other words, the trajectory described by the robot in the lake will converge asymptotically towards a cycle. Figure 3.20 has been obtained by taking

$$\epsilon = 0.05$$
$$[\mathbf{x}_0] = \begin{pmatrix} [-2-\epsilon, -2+\epsilon] \\ [2.3-\epsilon, 2.3+\epsilon] \\ [-2.3-\epsilon, -2.3+\epsilon] \end{pmatrix}$$

Remark 3.10. Note that to implement this example, we did not need the state variable τ, since its role is only to trigger the first guard of the hybrid system, which amounts to integrating ϕ_{q_0} for a duration T, as stated earlier.

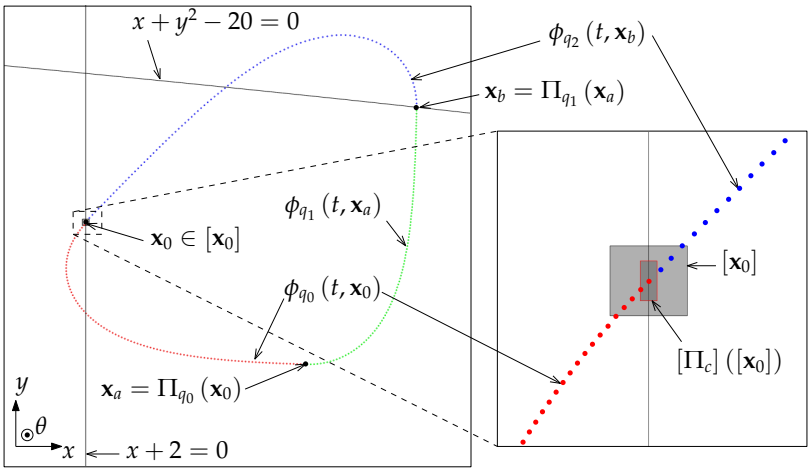


Figure 3.20: Proof of stability of the hybrid limit cycle

3.5 CHARACTERISATION OF STABILITY DOMAINS

As stated earlier, our method can only deal with small intervals. However, in underwater robotics, the uncertainties can be quite large, due to sensor precision, uncertain models, dead reckoning localisation... Therefore, our method might only be able to prove stability of a system inside a tiny initial box. Fortunately, other algorithms based on paving strategies can be used to approximate larger sets inside which the system is stable. We will not extend too much on these algorithms, since they are out of this thesis's scope. We will simply explain how they can be used jointly with our method, and for what applications.

3.5.1 *Characterising basins of attraction*

We introduced basins of attraction in Section 3.2.1.4. Consider a dynamical system with a stable fixed point $\bar{\mathbf{x}}$. Our algorithm can prove stability of the system inside a box $[\mathbf{x}_0] \ni \bar{\mathbf{x}}$. Denote by $\mathcal{B}_{\bar{\mathbf{x}}}$ the basin of attraction of $\bar{\mathbf{x}}$. Then $[\mathbf{x}_0] \subset \mathcal{B}_{\bar{\mathbf{x}}}$.

From there, choose any box $[\mathbf{x}]$ of $\mathbb{R}^n$. If a guaranteed integration method can prove that all the points initially in $[\mathbf{x}]$ end up in $[\mathbf{x}_0]$ after a certain amount of time, then this means that the system will also converge towards $\bar{\mathbf{x}}$ if initialised in $[\mathbf{x}]$. Therefore, by paving a subset of $\mathbb{R}^n$, one can find an inner approximation of $\mathcal{B}_{\bar{\mathbf{x}}}$. Note that a similar approach has been presented in [79], [80] to approximate *viability kernels* (we refer the reader to these papers for more details).

However, paving a subset of $\mathbb{R}^n$ is computationally expensive, and has exponential complexity in terms of dimensions. So special care must be taken in the way the paving is done, the boxes stored in memory...

Example 3.9. To illustrate this application, we will resume the example developed in Section 3.4.4 and find an inner approximation of the system's basin of attraction.

Since proving stability of the hybrid cycle comes down to proving stability of its associated hybrid Poincaré map, the basin of attraction of the latter, which is two-dimensional is a subset of the cycle's one, which is three-dimensional. We will thus concentrate on the basin of attraction of the hybrid Poincaré map, for visualisation purposes. We chose

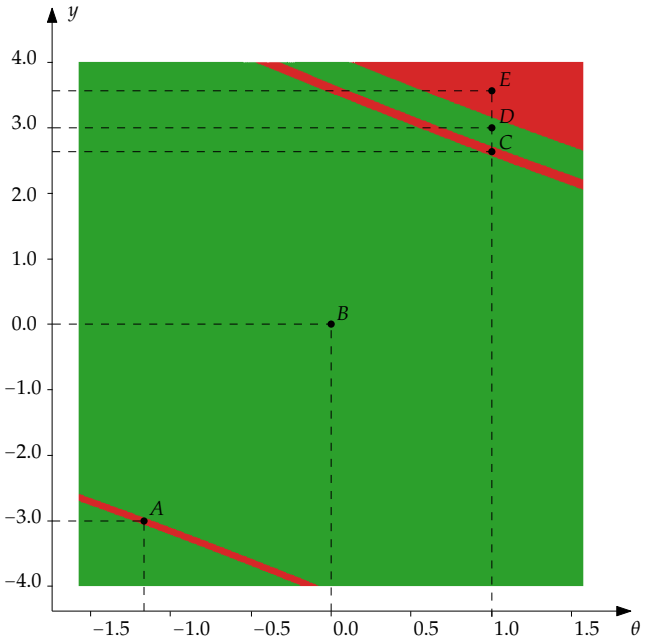


Figure 3.21: Basin of attraction (in green) of the hybrid Poincaré map

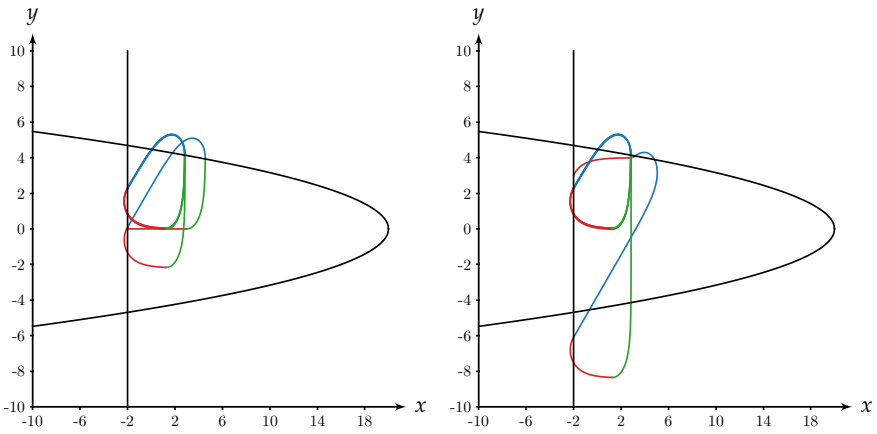
the plane $x = -2$ as the hybrid Poincaré map (this plane corresponds to the guard G_{e_2}). We chose the box $[\mathbf{u}] = \left([-2,-2],[-4,4],\left[\frac{-\pi}{2},\frac{\pi}{2}\right]\right)$ as the subset of $\mathbb{R}^n$ we wish to characterise.

In Figure 3.21, we displayed all the initial conditions leading the system to converge towards the box $[\mathbf{x}_0]$, such as defined in Section 3.4.4. Note that this figure is in the plane of the Poincaré section. To obtain it, we bisected the box $[\mathbf{u}]$ into sub-boxes $[\mathbf{y}]$, and we computed their images iteratively by the Poincaré map Π. If for a given k we had $\left[\mathbf{\Pi}_c^k\right]([\mathbf{y}]) \subset [\mathbf{x}_0]$, then we plotted the box in green, otherwise in red.

Remark 3.11. To obtain Figure 3.21, we used a "brute force" method, consisting in testing each box individually (about a million). However, it is possible to design a much more efficient algorithm using our centred form method to find a basin of attraction, e. g. testing one neighbourhood $[\mathbf{x}_0]$ and then finding all the initial conditions leading to $[\mathbf{x}_0]$, and iteratively assembling them to obtain a larger neighbourhood leading to stability.

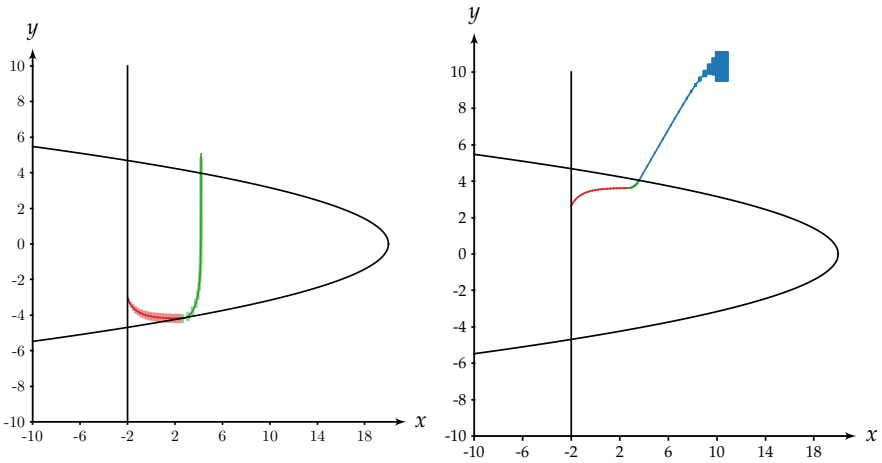
We plotted some remarkable trajectories initialised at the points A to E in Figures 3.22 and 3.23, to understand the meaning of the different areas than can be observed in Figure 3.21. Three different colours help visualise when the change of discrete state occur: the robot's trajectory while in state q_0 is displayed in red, q_1 in green and q_2 in blue.

Note that the trajectories represented in Figure 3.22 are examples where multiple iterations of the centred form are necessary to prove stability of the cycle (see Corollary 3.2).



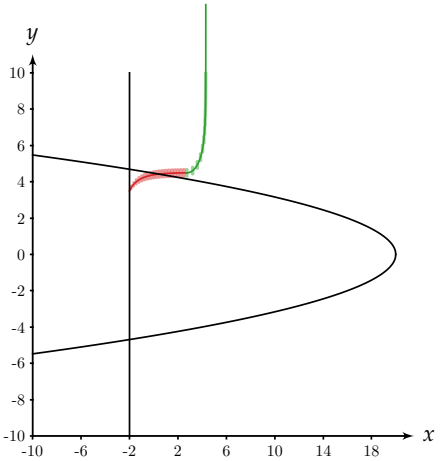
(a) Trajectory initialised at $\mathbf{x}_0 = B$ (b) Trajectory initialised at $\mathbf{x}_0 = D$

Figure 3.22: Examples of stable trajectories of a hybrid system



(a) Trajectory initialised at $\mathbf{x}_0 = A$: CAPD cannot compute the Poincaré map, because the green trajectory is initialised on its Poincaré section (parabola).

(b) Trajectory initialised at $\mathbf{x}_0 = C$: the system diverges because the controller is hesitating between turning left or right to follow its desired heading.



(c) Trajectory initialised at $\mathbf{x}_0 = E$: after leaving the first state q_0, the system will never cross the parabola.

Figure 3.23: Examples of unstable trajectories of a hybrid system

3.5.2 *Determining stability regions*

Another application of our method can be approximating the interior of a stability region of a system.

Definition 3.12 (Stability region)**.** Consider a discrete system of the form $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p})$ defined on an open subset $\mathcal{O}$ of $\mathbb{R}^n$, where $\mathbf{p} \in \mathbb{R}^m$ is a vector of uncertain parameters of the system. The *stability region* $\mathcal{S} \subset \mathcal{O}$ is the set of parameters $\mathbf{p}$ that makes the system converge asymptotically towards its fixed point $\bar{\mathbf{x}}$ (note that the latter may depend on $\mathbf{p}$)

$$\mathcal{S} = \left\{ \mathbf{p} \in \mathbb{R}^m \;\middle|\; \exists\, [\mathbf{x}_0] \ni \bar{\mathbf{x}} \subset \mathcal{O},\ \forall \mathbf{x}_0 \in [\mathbf{x}_0],\ \lim_{k\to\infty} \|\mathbf{x}_k - \bar{\mathbf{x}}\| = 0 \right\} \tag{3.102}$$

Finding a stability region is similar to approximating a basin of attraction, and the same kind of algorithms can be used. The parameter space must be paved, and for each $[\mathbf{p}]$ of the resulting paving $\mathcal{P}$, one must find a neighbourhood $[\mathbf{x}_0]$ such that the system is stable. Usually, width $([\mathbf{p}])$ must be small compared to width $([\mathbf{x}_0])$ for the stability algorithm to converge.

Example 3.10. To illustrate this application, we will resume Paragraph 3.4.2.3.1, and find the system's stability region. In this example, we will take as a parameter the robot's actual position, denoted by $\mathbf{x}$. The parameter space is therefore paved with boxes $[\mathbf{x}] = ([x], [y])$.

In Figure 3.24, we plotted in green the stability region of the system: if the real robot is placed inside the green area in a box $[\mathbf{x}]$, then it

is proven that there exists a neighbourhood $[\mathbf{x}_0] \supset [\mathbf{x}]$ such that the system will converge towards its fixed point, being any $\bar{\mathbf{x}} \in [\mathbf{x}]$.

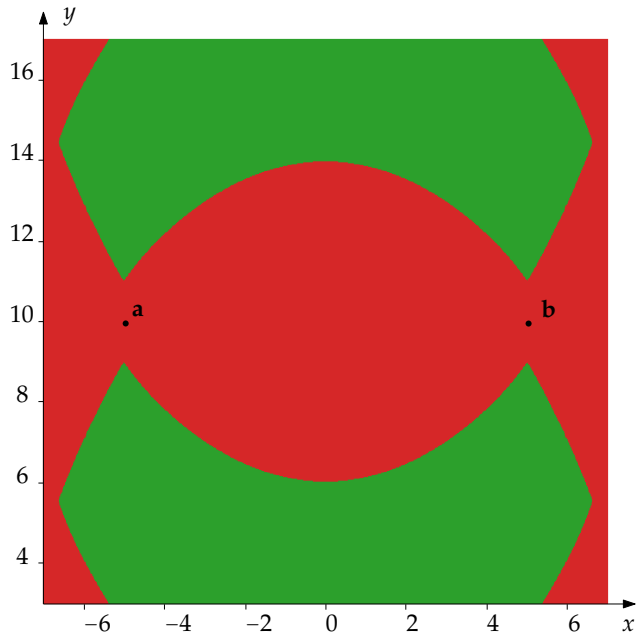


Figure 3.24: Stability region of the localisation system

Define $w_x = \text{width}([x])$ and $w_y = \text{width}([y])$. To compute this stability region, we chose the following parameters

$$\mathcal{O} = ([-7,7],[3,17]) \tag{3.103}$$

$$\forall [\mathbf{x}] \in \mathcal{P},\ \text{width}([\mathbf{x}]) \geq \epsilon \tag{3.104}$$

$$\forall [\mathbf{x}] \in \mathcal{P},\ [\mathbf{x}_0] = 0.5 \begin{pmatrix} [-\sqrt{w_x}, \sqrt{w_x}] \\ [-\sqrt{w_y}, \sqrt{w_y}] \end{pmatrix} \tag{3.105}$$

$$\epsilon = 0.03 \tag{3.106}$$

Remark 3.12. Similarly to the previous example, we used a "brute-force" method to compute the stability region since developing an optimal algorithm was not this thesis's goal.

3.6 CONCLUSION

This paragraph concludes the presentation of the methods we developed to prove stability of discrete-time and continuous-time uncertain systems. Using Poincaré maps, we have also shown that we were able to prove stability of a cyclic hybrid system. Our approach is based on the notions of *stability contractor* and *centred form*. It allows proving asymptotic stability of a system inside a given neighbourhood containing a fixed point (or an equilibrium state). The user chooses the

initial neighbourhood: if it is too large, the method may fail to prove stability, then the user simply needs to take a smaller neighbourhood. Note that our method either proves stability or cannot conclude: in particular, it cannot prove instability.

Contrary to the existing approaches allowing to prove stability of uncertain systems, such as the ones based on eigenvalues verification [102], the Jury criterion [49], or Lyapunov functions [26], ours allows proving asymptotic stability inside a chosen neighbourhood. The other methods, to the extent of our knowledge, only prove that there exists a neighbourhood inside which the system is stable, without specifying which one. Therefore, our method can be used to initialise algorithms approximating basins of attractions [26], [96] (see Section 3.5.1), stability regions (see Section 3.5.2) or backwards reachable sets [69].

Furthermore, interest of our method is its polynomial computational complexity. Once the Jacobian matrix of the system has been computed (once and for all), it merely consists in two interval function evaluations and two matrix multiplications. Therefore, handling high dimensional systems is possible, which would not be the case using an algorithm based on bisections, which has exponential complexity.

The main drawback of our approach is the obligation to deal with small intervals only. Indeed, the centred form performs better than a natural inclusion function only for small intervals. This explains why our method may fail when the initial neighbourhood is chosen too large.

4

PROVING FEASIBILITY OF A DOCKING MISSION : APPROACH BY REACHABILITY ANALYSIS

TABLE OF CONTENTS

4.1 INTRODUCTION

This chapter aims to present the second approach we developed to prove feasibility of a docking mission, which is based on *reachability analysis*. The *reachable set* of a dynamical system is the set of states that the system can attain when starting at a given position [5], [9], [21]. Consider a dynamical system initialised in $[\mathbf{x}_0]$ with a flow function $\phi(t, \mathbf{x})$. The reachable set of the system initialised in $[\mathbf{x}_0]$ over a duration T, denoted by $\mathcal{R}_T([\mathbf{x}_0])$ is defined by

$$\mathcal{R}_T([\mathbf{x}_0]) = \{\mathbf{x} \in \mathbb{R}^n \mid \exists t \in [0, T], \exists \mathbf{x}_0 \in [\mathbf{x}_0], \mathbf{x} = \phi(t, \mathbf{x}_0)\} \quad (4.1)$$

Reachable sets can then be used to prove that a system will avoid an obstacle, i. e. a set of forbidden states, or prove that a system will reach a certain area of space.

Now, the concept of proof requires the use of guaranteed methods, based on set-membership approaches, for example [4], [36], [95]. As we briefly explained in Section 2.6, approximating sets using boxes

can yield inner or outer approximations. While outer approximations of reachable sets can be obtained using guaranteed integration methods, inner approximations are much harder to find [38]. The outer approximation of the reachable set $\mathcal{R}$ of a system is exactly what is required to prove that the latter will avoid forbidden sets of states $\mathcal{O}$ (e. g. obstacles): if $\mathcal{R}_T([\mathbf{x}_0]) \cap \mathcal{O} = \emptyset$, the system will avoid them when initialised in $[\mathbf{x}_0]$ and while $t \leq T$. The inner approximation of a reachable set is not required to prove that a system will reach a specific region of space $\mathcal{T}$. If the enclosure of the state $[\mathbf{x}](t)$ is contained in $\mathcal{T}$ for a given $t \in [0, T]$, then the system is also proven to reach that specific region.

Therefore, we will be using guaranteed integration tools in this chapter (see Section 2.4), and in particular Lohner's algorithm (see Section 2.4.5).

It only remains for us to find a tool able to represent the reachable set of a system. In this thesis, we chose to use *tubes*, i. e. intervals of trajectories, that are inherently suitable for the task of enclosing a set of trajectories starting from an initial box $[\mathbf{x}_0]$.

In Section 4.2, we introduce the basics of tube arithmetic and the library Tubex [103]. In Section 4.3, we introduce the Lohner contractor, which can contract a tube using a differential constraint and the Lohner algorithm. Finally, in Sections 4.4 and 4.5, we illustrate this new contractor on a few examples.

4.2 INTERVALS AND TRAJECTORIES

4.2.1 *Tube arithmetic*

In the previous chapters, we introduced and used intervals to represent uncertainties. We saw that they could be used to enclose the set of solutions of an uncertain dynamical system: consider the following IVP:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \\ [\mathbf{x}](0) = [\mathbf{x}_0] \end{cases} \tag{4.2}$$

Then, using a guaranteed enclosure algorithm, one can find for any t a box $[\mathbf{x}](t)$ containing all the solutions of the IVP at that specific time. This evaluation could be done periodically, to obtain a sequence of boxes $[\mathbf{x}_k] = [\mathbf{x}](k \cdot dt)$, dt being the discretisation step. However, this sequence does not contain information about the system at any other time than $k \cdot dt$. Therefore, there is a need for a tool able to enclose the trajectories of a dynamical system over a certain period, and not only enclose its state at different times.

Remark 4.1.

1. In the previous chapters, we were calling *trajectories* the solutions of an IVP. Note that in this chapter, not all trajectories are solutions of an IVP. Mathematically speaking, a trajectory can be seen as a continuous function of time $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$;
2. To refer to trajectories as defined above, we will use the notation $\mathbf{x}(\cdot)$. In particular, $\mathbf{x}(t)$ is a point belonging to $\mathbf{x}(\cdot)$, i. e. the evaluation at a time t of the function $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$.

This leads us to the presentation of *tubes*, that were introduced in [63], and later adapted to constraint programming in an underwater robotics context [8], [67], [104].

Definition 4.1 (Tube)**.** A *tube* $[\mathbf{x}](\cdot) : \mathbb{R} \to \mathbb{R}^n$, defined on the time domain $[t_0, t_f]$, is an interval of trajectories $[\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$ such that for all $t \in [t_0, t_f]$, $\mathbf{x}^-(t) \leq \mathbf{x}^+(t)$.

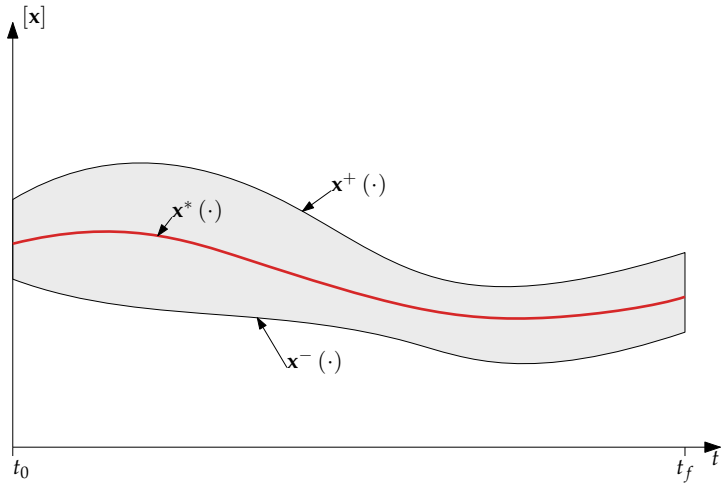


Figure 4.1: Illustration of a one-dimensional tube

Usual operators can be defined over tubes: let $[\mathbf{x}](\cdot)$ and $[\mathbf{y}](\cdot)$ be two tubes and choose an operator $\diamond \in \{+, -, \cdot, /\}$. The resulting binary operation between the two tubes is defined as follows

$$[\mathbf{x}](\cdot) \diamond [\mathbf{y}](\cdot) = [\{\mathbf{x}(\cdot) \diamond \mathbf{y}(\cdot) \in \mathbb{R}^n \mid \mathbf{x}(\cdot) \in [\mathbf{x}](\cdot), \mathbf{y}(\cdot) \in [\mathbf{y}](\cdot)\}] \tag{4.3}$$

Similarly, let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a function. The smallest image by $\mathbf{f}$ of a tube $[\mathbf{x}](\cdot)$ is given by

$$\mathbf{f}([\mathbf{x}](\cdot)) = [\{\mathbf{f}(\mathbf{x}(\cdot)) \mid \mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)\}] \tag{4.4}$$

This short introduction to tube arithmetic will be sufficient for the rest of this chapter. However, more operations can be performed using them. We refer the reader to [105] for more details.

4.2.2 *Tube contraction*

In Section 2.5, we gave a brief introduction to interval contractors, which aim at contracting a box $[\mathbf{x}]$ according to a set of constraints formalising a problem. This approach has been extended to tubes (see [8], [104]).

Definition 4.2. Consider a constraint $\mathcal{L}$, and a tube $[\mathbf{x}](\cdot)$. A contractor for $\mathcal{L}$ over $[\mathbf{x}](\cdot)$ is an operator $\mathcal{C} : \mathbb{IR}^n \to \mathbb{IR}^n$ such that

$$\forall t,\ \mathcal{C}\left([\mathbf{x}](\cdot)\right)(t) \subseteq [\mathbf{x}](t) \tag{4.5}$$

(monotonicity)

$$\begin{pmatrix} \mathcal{L}(\mathbf{x}(\cdot)) \\ \mathbf{x}(\cdot) \in [\mathbf{x}](\cdot) \end{pmatrix} \implies \mathbf{x}(\cdot) \in \mathcal{C}\left([\mathbf{x}](\cdot)\right) \tag{4.6}$$

(completeness)

Equation (4.5) states that the contracted tube remains in the initial tube, while Equation (4.6) ensures that no trajectory verifying the constraint is removed from the initial tube.

Unlike intervals and boxes, tubes contain time information. Therefore, two types of contractors can be defined. *Static contractors* are based on static constraints, i. e. that contract the tube independently of time (equivalent to interval contractors introduced in Section 2.5). *Temporal contractors* impose time constraints, commonly differential constraints from a differential equation modelling a dynamical system.

Example 4.1. In [104], the authors proposed the so-called $\mathcal{C}_{\frac{d}{dt}}$ contractor, which contracts a pair of tubes $[\mathbf{x}](\cdot)$ and $[\mathbf{v}](\cdot)$, the former containing the trajectories of a system, and the latter their derivatives.

The use case of this contractor is the following: consider a robot evolving in its environment and recording its position $\mathbf{x}$ and its velocity $\mathbf{v}$. The latter can therefore be enclosed in two tubes $[\mathbf{x}](\cdot)$ and $[\mathbf{v}](\cdot)$ to account for measurement errors. Both tubes can then be contracted by considering the differential constraint linking them ($\dot{\mathbf{x}} = \mathbf{v}$). Using a first-order Euler integration scheme over an arbitrary period dt, they built the following contractor

$$[\mathbf{x}](t+dt) = [\mathbf{x}](t+dt) \cap \left([\mathbf{x}](t) + dt \cdot [\mathbf{v}]([t, t+dt])\right) \tag{4.7}$$

4.2.3 *Implementation : the Tubex library*

Now that we introduced the basics of tube arithmetic, we will focus on their implementation in a computer program, and more precisely in the Tubex library. The choice that has been made is to represent

trajectories through a sequence of *slices*, as represented in Figure 4.2. This slicing results from the sampling of the tube over time: consider a tube $[\mathbf{x}](\cdot) = [\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$ defined over a time interval $[t_0, t_f]$. Let $\delta > 0$ be the sampling period. Then, $[\mathbf{x}](\cdot)$ is outer approximated by a sequence of slices denoted by $[\mathbf{x}](k)$ such that

$$[\mathbf{x}](k) = [k\delta, (k+1)\delta] \times \left[\underline{\mathbf{x}^-}(\tau_k), \overline{\mathbf{x}^+}(\tau_k)\right] \tag{4.8}$$

where $\tau_k \in]k\delta, (k+1)\delta[$ and $\underline{\mathbf{x}^-}(t)$ and $\overline{\mathbf{x}^+}(t)$ are step functions such that

$$\forall t \in [t_0, t_f], \underline{\mathbf{x}^-}(t) \leq \mathbf{x}^-(t) \leq \mathbf{x}^+(t) \leq \overline{\mathbf{x}^+}(t) \tag{4.9}$$

Note that when $\tau_k = k\delta$, the tube is evaluated using the slices indexed $k-1$ and k: $[\mathbf{x}](\tau_k) = [\mathbf{x}](k-1) \cap [\mathbf{x}](k)$. This quantity is then called the *input gate* of slice k and the *output gate* of slice $k-1$.

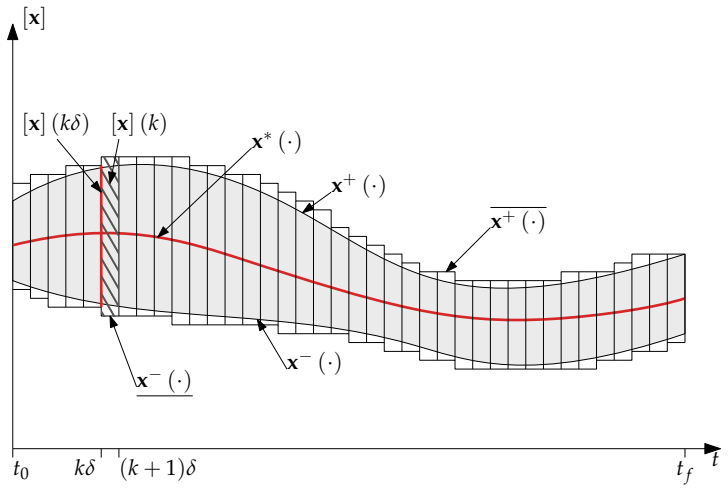


Figure 4.2: Representation of a tube and its sliced counterpart

4.2.4 *Summary of the notations*

For the sake of clarity, we recall below the main notations used later in this chapter:

- $[\mathbf{x}](\cdot) \in \mathbb{R} \times \mathbb{IR}^n$ is a tube, implicitly defined on a time interval $[t_0, t_f]$;
- $[\mathbf{x}](t) \subset \mathbb{R}^n$ is a box, i. e. an evaluation of the tube at time $t \in [t_0, t_f]$, i. e. an enclosure for the state of the system at time t;
- $[\mathbf{x}](k) \in \mathbb{R} \times \mathbb{IR}^n$ is the k^{th} slice of the tube, defined over $[t_k, t_{k+1}]$ where for all k, $t_k = k\delta \leq t_f$, δ being the sampling period of the tube;
- $[\mathbf{x}](t_k) \subset \mathbb{R}^n$ is a box, and corresponds to the input gate of the k^{th} slice, and the output gate of the $(k-1)^{\text{th}}$ slice.

4.3 TEMPORAL CONTRACTORS : DIFFERENTIAL CONSTRAINTS

This section will present the so-called *Picard contractor*, which is a temporal contractor for the differential constraint $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. We will then introduce this chapter's main contribution, i. e. the *Lohner contractor*, which is also a contractor for such a differential constraint. The Lohner contractor's main advantage is its smaller wrapping effect compared to that of the Picard contractor when the initial tube $[\mathbf{x}](\cdot)$ is thin.

4.3.1 *Picard contractor*

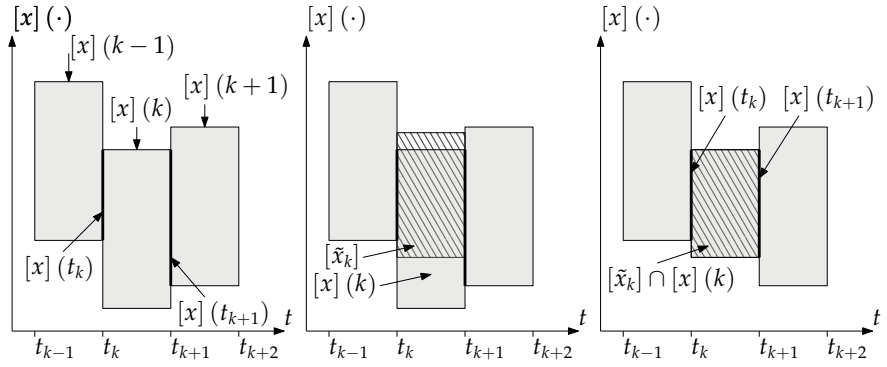
This contractor has been implemented in Tubex, but is nowhere to be found in the literature. The rest of this section is therefore based on what we understood from its implementation.

To the extent of our knowledge, the Picard contractor, implemented in Tubex, is the only temporal contractor able to deal with a differential constraint of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. This contractor is loosely based on Algorithm 1, used to compute global enclosures in Lohner's algorithm.

Consider a tube $[\mathbf{x}](\cdot)$ defined on a time interval $[t_0, t_f]$, and a slice $[\mathbf{x}](k)$. Denote by $t_k = k\delta$ the sampling times of the tube. Using Algorithm 1 and taking as input for the algorithm the input gate of $[\mathbf{x}](k)$ denoted by $[\mathbf{x}](t_k)$, the contractor can compute a global enclosure $[\tilde{\mathbf{x}}_k]$ for $[\mathbf{x}](\cdot)$ over the time interval $[t_k, t_{k+1}]$. In other words, we have

$$\forall t \in [t_k, t_{k+1}], \ [\mathbf{x}](t) \subset [\tilde{\mathbf{x}}_k] \tag{4.10}$$

Therefore, since both the slice $[\mathbf{x}](k)$ and the global enclosure $[\tilde{\mathbf{x}}_k]$ contain all the trajectories of the system leaving from the input gate $[\mathbf{x}](t_k)$, so does their intersection, as illustrated in Figure 4.3.



(a) Initial tube

(b) Computation of the global enclosure $[\tilde{\mathbf{x}}_k]$ using the input gate $[\mathbf{x}](t_k)$

(c) Result of the contraction of the k^{th} slice of the tube

Figure 4.3: Contracting one slice of a tube using the Picard contractor

Proposition 4.1. *Consider a dynamical system described by* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$*, and a tube of trajectories* $[\mathbf{x}](\cdot)$ *defined over* $[t_0, t_f]$*. Discretise the latter with a period* δ *and denote by* $t_k = k \cdot \delta$ *the resulting discretisation times. Then the following operator is a contractor for* $[\mathbf{x}](\cdot)$ *over* $[t_0, t_f]$

$$\forall t \in [t_k, t_{k+1}], \forall k \leq \frac{t_f}{\delta}, \mathcal{C}_{Picard}([\mathbf{x}](t)) = [\mathbf{x}](t) \cap [\tilde{\mathbf{x}}_k] \tag{4.11}$$

where

$$[\tilde{\mathbf{x}}_k] \supset \left[\left\{\phi_{[\mathbf{x}](t_k)}(t) \,\middle|\, t \in [t_k, t_{k+1}]\right\}\right] \tag{4.12}$$

is the global enclosure of the system's trajectories over $[t_k, t_{k+1}]$*.*

Proof. The *monotonicity* property (Equation (4.5)) comes from Equation (4.11)

$$\begin{aligned}\forall t \in [t_k, t_{k+1}], \forall k \leq \frac{t_f}{\delta}, \mathcal{C}_{Picard}([\mathbf{x}](t)) &= [\mathbf{x}](t) \cap [\tilde{\mathbf{x}}_k] \\ &\subseteq [\mathbf{x}](t)\end{aligned}$$

The *completeness* property (Equation (4.6)) comes from the Banach fixed-point theorem: if Algorithm 1 returns, this means that for all $\mathbf{x}_k \in [\mathbf{x}](t_k)$, there exists a trajectory defined over $[t_k, t_{k+1}]$ contained in $[\tilde{\mathbf{x}}_k]$. Therefore, all the trajectories starting from the input gate $[\mathbf{x}](t_k)$ are contained in $[\tilde{\mathbf{x}}_k]$, and no trajectory satisfying the differential constraint is removed from the initial tube. ■

This contractor has one major drawback: the global enclosure $[\tilde{\mathbf{x}}_k]$ encloses all the system's states in the time interval $[t_k, t_{k+1}]$. Therefore, the $(k+1)^{\text{th}}$ slice's input gate not only depends on the state at time t_{k+1}, but also on all the previous states used to compute the global enclosure. This results in an over-approximation of the input gate $[\mathbf{x}](t_{k+1})$, and thus in increasing wrapping effect over time.

Example 4.2. Consider the one-dimensional system described by Equation (4.13).

$$\dot{x} = -\sin(x) \tag{4.13}$$

Let us take $[x_0] = [0.9, 1.1]$ as an initial state, thanks to which we can initialise a tube $[x](\cdot)$ as follows

$$[x](t) = \begin{cases} [x_0] & \text{if } t = 0; \\ [-\infty, \infty] & \text{otherwise.} \end{cases} \tag{4.14}$$

In Figure 4.4, we plotted the tube $\mathcal{C}_{Picard}([x](\cdot))$ over the time interval $[0, 10]$. The tube itself is outer approximated by its slices of width $\delta = 0.1$, and a few trajectories initialised in $[x_0]$ are plotted in blue.

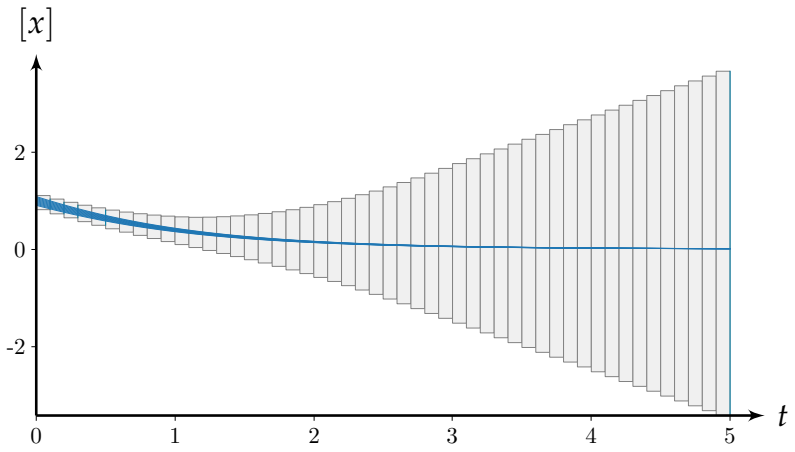


Figure 4.4: One-dimensional tube contracted by $\mathcal{C}_{Picard}$

One can notice that the Picard contractor manages to contract the first slices correctly, but gradually loses contraction power, which is due to the wrapping effect introduced by this method. Therefore, this contractor should only be used for short-term integration.

Remark 4.2. In this section, we only describe the forward step of contraction, i. e. starting from the first slice and contracting the tube in the direction of increasing time. However, once this forward step is done, a backward step can be executed to back-propagate the possible extra information through the tube. In this example, it would not be useful, because no extra information is available (all the slices are infinitely large). However, suppose we were contracting a tube that had already been through a contraction step, e. g. by a static contractor. In that case, this information could be propagated back and forth through the tube via forward-backwards contractor composition. We refer the reader to [50], [105] for more details about contractor composition strategies.

4.3.2 *Lohner contractor*

To overcome the Picard contractor's major drawback, highlighted in the previous section, we decided to implement a temporal contractor for a differential constraint of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ based on Lohner's algorithm (see Algorithm 2). Let us recall the working principle of the latter:

1. given an initial enclosure of the state $[\mathbf{x}_0]$, the algorithm is capable of computing successive enclosures $[\mathbf{x}_k]$, enclosing the state of the system at time $t_k = k\delta$;
2. for all $k \geq 0$, the enclosure $[\mathbf{x}_k]$ is represented by a cuboid (see Section 2.4.4.3): $[\mathbf{x}_k] = \hat{\mathbf{x}}_k + \mathbf{B}_k \cdot [\mathbf{r}_k]$. Both $[\mathbf{x}_k]$ and $[\mathbf{r}_k]$ represent

the uncertainties of the enclosures, respectively in the canonical basis and in the one defined by the orthogonal matrix $\mathbf{B}_k$;

3. to compute $[\mathbf{x}_{k+1}]$ from $[\mathbf{x}_k]$, the algorithm estimates the global enclosure $[\tilde{\mathbf{x}}_k]$, such that for all $t \in [t_k, t_{k+1}]$ and for all $\mathbf{x}_k \in [\mathbf{x}_k]$, $\phi(t - t_k, \mathbf{x}_k) \in [\tilde{\mathbf{x}}_k]$. The latter is estimated using Algorithm 1.

Now, consider a tube $[\mathbf{x}](\cdot)$ enclosing the state of a system defined by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ over a time interval $[t_0, t_f]$. Assume that the tube is represented in the computer's memory by slices of width δ. Then,

1. $[\mathbf{x}](t_0)$ can be seen as the input gate of the slice $[\mathbf{x}](0)$ and the initial condition of the system;
2. define $t_k = k\delta$; then $[\mathbf{x}](t_k)$ can be interpreted as the input gate of the k^{th} slice, the output gate of the $(k-1)^{\text{th}}$ slice, and the enclosure of the state at time t_k;
3. the slice $[\mathbf{x}](k)$ can be interpreted as a slice of the tube, containing all the trajectories over the time interval $[t_k, t_{k+1}]$, but also as a global enclosure of the system over the same time interval.

We will now explain how the tube $[\mathbf{x}](\cdot)$ can be contracted according to the differential constraint using Lohner's algorithm. First, assume that we initialised Lohner's algorithm with the initial state $[\mathbf{x}_0] = [\mathbf{x}](t_0)$, and assume that we reached the iteration k, i. e. the time $t_k = k\delta$. Then,

1. we know from the initial tube that all the trajectories are contained inside the slice $[\mathbf{x}](k)$, and from Lohner's algorithm that the trajectories verifying the differential equation are contained inside $[\tilde{\mathbf{x}}_k]$;
2. we know from the initial tube that the system's state at time t_{k+1} is enclosed in the output gate $[\mathbf{x}_{k+1}]$, and from Lohner's algorithm that the system evolving during a duration δ from $[\mathbf{x}_k]$ is contained inside $[\mathbf{x}_{k+1}]$;
3. Finally, we know that $[\mathbf{x}_k] = \hat{\mathbf{x}}_k + \mathbf{B}_k \cdot [\mathbf{r}_k]$.

Algorithm 4 summarises the contractions performed on the tube and on the boxes used inside Lohner's algorithm (namely $[\mathbf{x}_k]$ and $[\mathbf{r}_k]$), and Figure 4.5 illustrates the working principle of the algorithm.

Proposition 4.2. *Consider a dynamical system described by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, and a tube of trajectories $[\mathbf{x}](\cdot)$ defined over $[t_0, t_f]$. Then the operator $\mathcal{C}_{Lohner}$ is a contractor for $[\mathbf{x}](\cdot)$ over $[t_0, t_f]$.*

Algorithm 4 $\mathcal{C}_{Lohner}$ `(inout : [x](·))`

Initialisation:
$[\mathbf{x}_0] \leftarrow [\mathbf{x}](t_0)$
$\hat{\mathbf{x}}_0 \leftarrow \text{mid}([\mathbf{x}_0])$
$[\mathbf{z}_0] \leftarrow [\mathbf{x}_0] - \hat{\mathbf{x}}_0$
$\mathbf{m}_0 \leftarrow \text{mid}([\mathbf{z}_0]) = \mathbf{0}$
$[\mathbf{r}_0] \leftarrow [\mathbf{z}_0] - \mathbf{m}_0 = [\mathbf{z}_0]$
$\mathbf{B}_0 = \mathbf{I}$
Main loop:
for $k = 0$ **to** $\frac{t_f}{\delta}$ **do**
  Lohner integration: *see Algorithm 2*
  $[\mathbf{A}_k] \leftarrow \mathbf{I} + h\,[\mathbf{J_f}]([\mathbf{x}_k])$
  $[\tilde{\mathbf{x}}_k] \leftarrow$`GlobalEnclosure`$([\mathbf{x}_k])$ *see Algorithm 1**
  $[\mathbf{z}_{k+1}] \leftarrow \frac{\delta^2}{2}\,[\mathbf{J_f}]([\tilde{\mathbf{x}}_k])\,[\mathbf{f}]([\tilde{\mathbf{x}}_k])$
  $\mathbf{m}_{k+1} \leftarrow \text{mid}([\mathbf{z}_{k+1}])$
  $\mathbf{B}_{k+1} \leftarrow \text{Qr}(\text{mid}([\mathbf{A}_k]\,\mathbf{B}_k))$ **
  $[\mathbf{r}_{k+1}] \leftarrow \left(\mathbf{B}_{k+1}^{-1}\,[\mathbf{A}_k]\,\mathbf{B}_k\right)[\mathbf{r}_k] + \mathbf{B}_{k+1}^{-1}([\mathbf{z}_{k+1}] - \mathbf{m}_{k+1})$
  $\hat{\mathbf{x}}_{k+1} \leftarrow \hat{\mathbf{x}}_k + \delta\mathbf{f}(\hat{\mathbf{x}}_k) + \mathbf{m}_{k+1}$
  $[\mathbf{x}_{k+1}] \leftarrow \hat{\mathbf{x}}_{k+1} + \mathbf{B}_{k+1}\,[\mathbf{r}_{k+1}]$
  Tube contraction:
  $[\mathbf{x}_{k+1}] \leftarrow [\mathbf{x}_{k+1}] \cap [\mathbf{x}](t_{k+1})$
  $\hat{\mathbf{x}}_{k+1} \leftarrow \text{mid}([\mathbf{x}_{k+1}])$
  $[\mathbf{r}_{k+1}] \leftarrow [\mathbf{r}_{k+1}] \cap \left(\mathbf{B}_{k+1}^{-1} \cdot ([\mathbf{x}_{k+1}] - \hat{\mathbf{x}}_{k+1})\right)$
  $$[\mathbf{x}](t) = \begin{cases} [\mathbf{x}](t) \cap [\tilde{\mathbf{x}}_k] & \text{if } t \in [t_k, t_{k+1}[\,; \\ [\mathbf{x}](t) \cap [\mathbf{x}_{k+1}] & \text{if } t = t_{k+1}. \end{cases}$$
end for

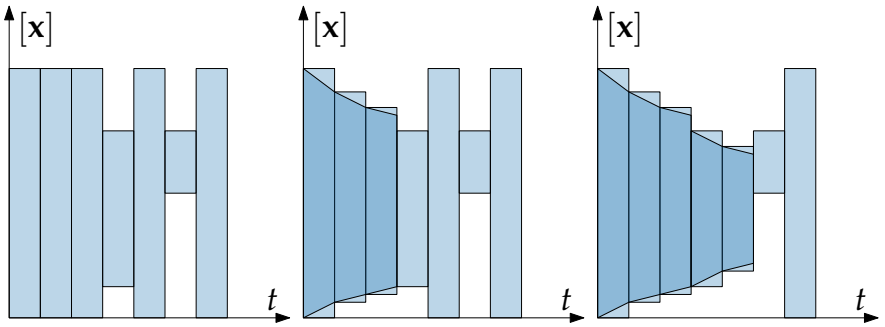
* for the sake of simplicity, we consider that h does not change because of `GlobalEnclosure`
** Qr() returns the orthogonal part of the QR factorization

Proof. The *consistency* property comes from the 23rd line of the algorithm: for all $t \in [t_0, t_f]$, we have

$$\begin{aligned} \mathcal{C}_{Lohner}([\mathbf{x}](\cdot))(t) &= [\mathbf{x}](t) \cap \begin{cases} [\tilde{\mathbf{x}}_k] & \text{if } t \neq t_k; \\ [\mathbf{x}_k] & \text{if } t = t_k. \end{cases} \\ &\subset [\mathbf{x}](t) \end{aligned}$$

The *completeness* property comes from the 20th and 23rd lines of the algorithm:

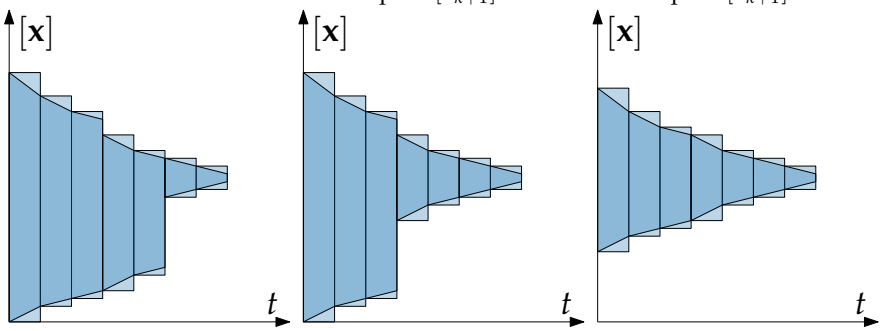
- the state at time t_{k+1} is contained in $[\mathbf{x}_{k+1}]$ according to Lohner's algorithm and in $[\mathbf{x}](t_{k+1})$ according to the initial tube. Therefore no solution verifying both the differential constraint and the initial enclosure is removed;



(a) Initial tube $[\mathbf{x}](\cdot)$

(b) Contraction of the three first slices: the contractor considers $[\mathbf{x}_k] \cap [\mathbf{x}](t_k)$ to compute $[\mathbf{x}_{k+1}]$

(c) Contraction of the two next slices: the contractor considers $[\mathbf{x}_k] \cap [\mathbf{x}](t_k)$ to compute $[\mathbf{x}_{k+1}]$



(d) Contraction of the remaining slices in forward mode

(e) Beginning of backward mode

(f) Contracted tube $\mathcal{C}_{Lohner}([\mathbf{x}](\cdot))$

Figure 4.5: Illustration of the principle of $\mathcal{C}_{Lohner}$

- note that the 22$^{\text{nd}}$ line simply transposes the result of this first contraction to the cuboid, which is a tighter enclosure for the state at time t_{k+1}; and that the 21$^{\text{st}}$ line is there to keep $[\mathbf{r}_{k+1}]$ centred on the origin;
- the system's trajectories over the time interval $[t_k, t_{k+1}]$ are contained in $[\mathbf{x}](k)$, i. e. the k^{th} slice of the tube, and in $[\tilde{\mathbf{x}}_k]$. Therefore, no trajectory satisfying both the differential constraint and the initial enclosure is removed.

■

Remark 4.3. Note that one of the main differences between the Picard and Lohner contractors is that the former uses the input gate $[\mathbf{x}](t_k) \cap [\tilde{\mathbf{x}}_{k-1}]$ to compute the global enclosure $[\tilde{\mathbf{x}}_k]$, while the latter uses $[\mathbf{x}](t_k) \cap [\mathbf{x}_k]$. This combined with a sharper integration method yields better results.

Example 4.3. In this example, we will resume Example 4.2 to compare the results of both methods. Figure 4.6 has been obtained using $\mathcal{C}_{Lohner}$ instead of $\mathcal{C}_{Picard}$.

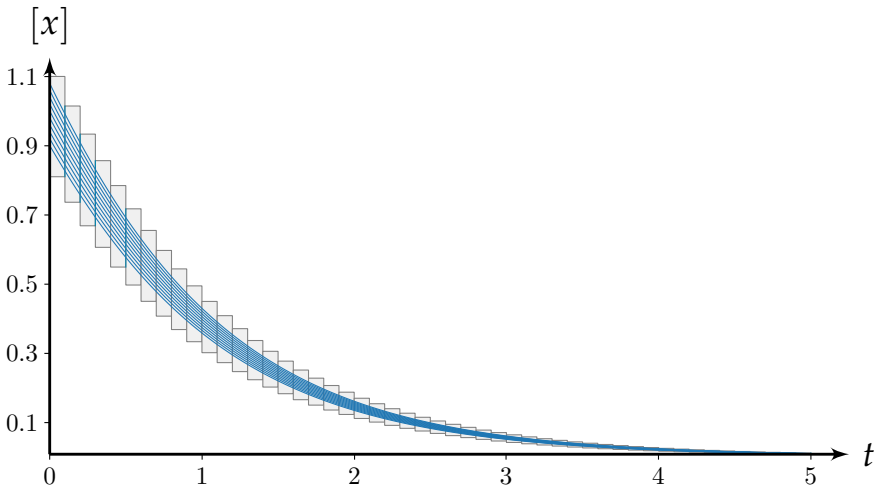


Figure 4.6: One-dimensional tube contracted by $\mathcal{C}_{Lohner}$

The figure speaks for itself: the Lohner contractor manages to deal with far longer time integration than the Picard contractor, and avoids adding too much wrapping effect, allowing the tube to converge towards 0.

This last example demonstrates that the Lohner contractor can be much more contracting than the Picard one. However, it has one major disadvantage: just like Lohner's algorithm, it can only deal with relatively small boxes, i. e. thin tubes. Otherwise, its contraction power will be reduced to nothing.

Remark 4.4. We only described the forward step of the Lohner contractor, but note that it is also possible to similarly implement the backward step, by integrating backwards in time from the output gate of the last slice of the tube.

4.4 TUBES AND REACHABILITY ANALYSIS

This section is intended to illustrate the use of the Lohner contractor as presented earlier in a robotics context. In particular, it can be used to study reachability of a system.

As in the previous chapter, by "system" we mean a mathematical model for the robot, its sensors, actuators and embedded algorithms.

As stated earlier in Section 4.2.3, tubes, when implemented as slices, consist in an outer approximation of the trajectories of a system, i. e. of its possible states over a time interval $[0, T]$. In other words, consider the tube $[\mathbf{x}](\cdot)$ defined over $[0, T]$, it follows that

$$\mathcal{R}_T([\mathbf{x}_0]) \subset [\mathbf{x}](\cdot) \tag{4.15}$$

To illustrate this application, consider a tank robot modelled by Equation (4.16) and initialised in the box $[\mathbf{x}_0]$. The robot follows the

vector field defined by Equation (4.18) thanks to the controller given by Equation (4.17). This vector field is repulsive in the surroundings of an obstacle placed at $\mathbf{x}_o$ and attracts the robot towards a point $\mathbf{x}_t$.

$$\dot{\mathbf{x}} = \begin{pmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ 10 \cdot \sin(\theta_d - \theta) \\ 1.5 \cdot (v_d - v) \end{pmatrix} \tag{4.16}$$

$$\begin{pmatrix} \theta_d \\ v_d \end{pmatrix} = \begin{pmatrix} \text{atan2}(c_y, c_x) \\ \|\mathbf{c}\| \end{pmatrix} \tag{4.17}$$

$$\mathbf{c} = (c_x, c_y) = \mathbf{x}_t - \mathbf{x} - \frac{\mathbf{x}_o - \mathbf{x}}{\|\mathbf{x}_o - \mathbf{x}\|^2} + \begin{pmatrix} \frac{y - y_o}{\|\mathbf{x}_o - \mathbf{x}\|^2} \\ \frac{-x + x_o}{\|\mathbf{x}_o - \mathbf{x}\|^2} \end{pmatrix} \tag{4.18}$$

To obtain the reachability set of the robot, we initialise the tube of trajectories as follows

$$[\mathbf{x}](t) = \begin{cases} [\mathbf{x}_0] & \text{if } t = 0; \\ [-\infty, \infty]^{\times 4} & \text{otherwise.} \end{cases} \tag{4.19}$$

We arbitrarily set the time interval of the tube to $[0, 2.1]$. Then, we contract this initial tube using $\mathcal{C}_{Lohner}$. Ideally, we would like the robot to reach a neighbourhood of its target defined by

$$[\mathbf{x}_t] = \mathbf{x}_t + ([-\epsilon, \epsilon], [-\epsilon, \epsilon]) \tag{4.20}$$

where $\epsilon = 0.1$. Figure 4.7 is the result of this contraction.

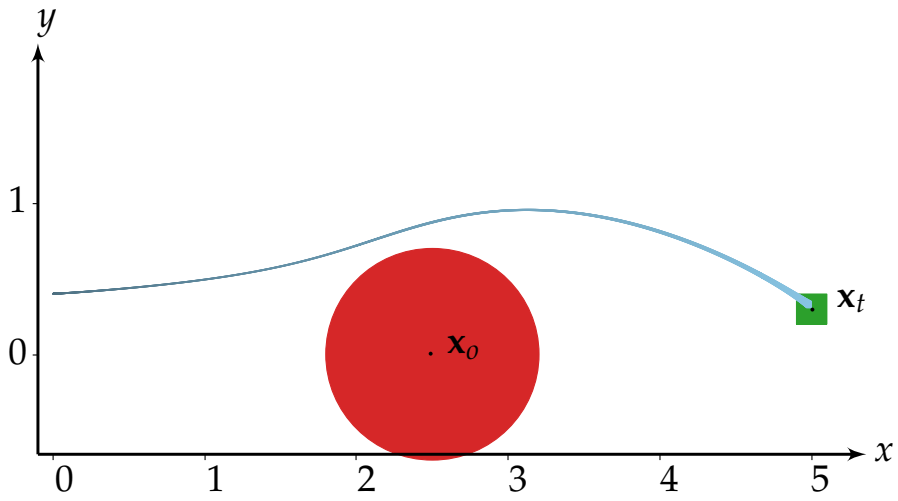


Figure 4.7: Reachability analysis of a robot driven by a potential field

From this figure, we can deduce that when the robot is initialised in $[\mathbf{x}_0]$, its controller enables it to reach the green box symbolising $[\mathbf{x}_t]$ while avoiding the obstacle, symbolised by the red disk.

4.5 APPROXIMATION OF CAPTURE TUBES

In this section, we will give another example of an application for the Lohner contractor. This example is less related to the docking problem than the previous one, but illustrates the Lohner contractor's adequacy to solve specific problems.

In [52], the authors present a method to compute *capture tubes* (also called *positive invariant tubes*). These tubes are a way of formalising positive invariant sets of dynamical systems [9].

Definition 4.3 (Capture tube)**.** Consider a continuous-time system with flow function ϕ. A tube $[\mathbf{x}](\cdot)$ is said to be a *capture tube* for the system if

$$\forall t > 0,\ \mathbf{x}(t_0) \in [\mathbf{x}](t_0) \implies \mathbf{x}(t_0 + t) \in [\mathbf{x}](t_0 + t) \tag{4.21}$$

In other words, a tube $[\mathbf{x}](\cdot)$ is a capture tube if all the trajectories entering it at a given time t_0 remain inside it.

The method presented in [52] consists in choosing an arbitrary candidate capture tube for the system $[\mathbf{c}^*](\cdot)$, and then checking whether some trajectories leave the latter using contractor programming and guaranteed integration algorithms. This approach allows computing an outer approximation $[\mathbf{c}^+](\cdot)$ of the real capture tube $[\mathbf{c}](\cdot)$, as illustrated in Figure 4.8.

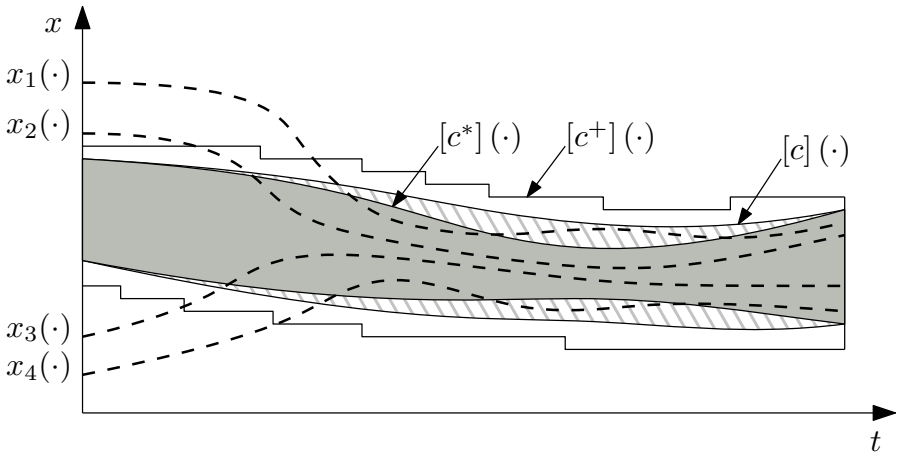
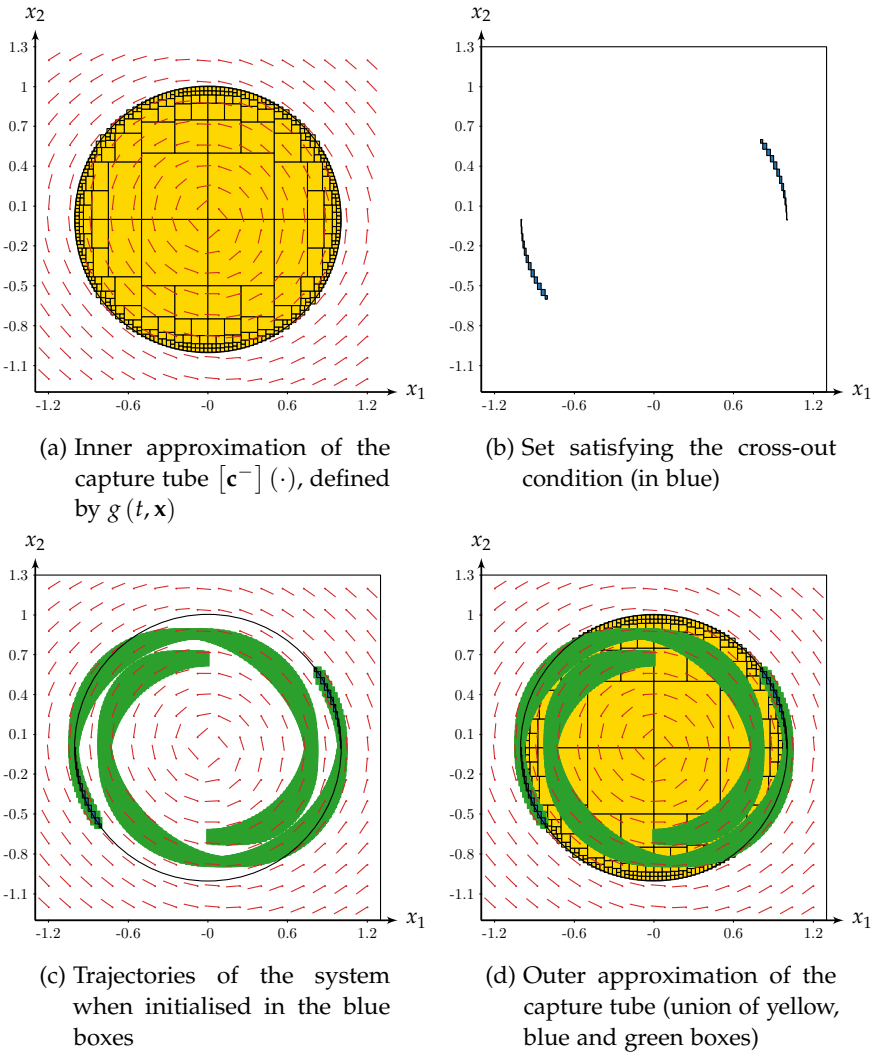


Figure 4.8: Illustration of a capture tube, its inner and its outer approximation

The example given in [52] is the following:

1. Consider a simple pendulum, such as described by Equation (2.41) with $d = 0.15$ as damping parameter;
2. Define the candidate capture tube $[\mathbf{c}^*](\cdot)$ by the function

$$g(t, \mathbf{x}) = x_1^2 + x_2^2 - 1 \tag{4.22}$$



(a) Inner approximation of the capture tube $[\mathbf{c}^-](\cdot)$, defined by $g(t, \mathbf{x})$

(b) Set satisfying the cross-out condition (in blue)

(c) Trajectories of the system when initialised in the blue boxes

(d) Outer approximation of the capture tube (union of yellow, blue and green boxes)

Figure 4.9: Approximation of $[\mathbf{c}^+](\cdot)$ using the method described in [52]

This tube does not depend on time, and corresponds to a disk of radius 1 m, i. e. for all $t > 0$, $g(t, [\mathbf{c}^*](t)) \leq 0$.

3. Define the cross-out condition contractor, which allows finding the boxes inside which the system leaves the disk aforementioned, i. e. the boxes on the border of the disk where the scalar product between the vector field and the gradient of the border is positive

$$\mathcal{C}_{\textit{cross-out}}([\mathbf{x}]) = \left[\left\{\mathbf{x} \in [\mathbf{x}] \,\middle|\, g(t, \mathbf{x}) = 0 \wedge \mathbf{f}(\mathbf{x})^{\mathrm{T}} \cdot \frac{\partial g}{\partial \mathbf{x}}(t, \mathbf{x}) > 0\right\}\right] \tag{4.23}$$

4. Using a paving algorithm, coupled to $\mathcal{C}_{\textit{cross-out}}$, find a subpaving of boxes $[\mathbf{x}](0)$ inside which the cross-out constraint is verified

5. Using a guaranteed integration algorithm (in [52], the authors use the DynIbex library [27]), integrate the box $[\mathbf{x}](0)$ during an arbitrary time T, and check whether the resulting box $[\mathbf{x}](T)$ is inside the disk

$$\mathcal{C}_{disk}([\mathbf{x}]) = [\{\mathbf{x} \in [\mathbf{x}] \mid g(t, \mathbf{x}) > 0\}] \tag{4.24}$$

6. If $[\mathbf{x}](T)$ is in the disk, i. e. $\mathcal{C}_{disk}([\mathbf{x}](T)) = \emptyset$, then this means that if the system leaves the disk through the box $[\mathbf{0}]$, it will re-enter the latter after a duration T, and the tube $[\mathbf{x}](\cdot)$ is a part of the outer approximation $[\mathbf{c}^+](\cdot)$.

7. Otherwise, use smaller boxes $[\mathbf{x}](0)$ through bisections and contractions by $\mathcal{C}_{cross\text{-}out}$.

We reproduced the results obtained in [52] in Figure 4.9.

Below, we propose a method based on the Lohner contractor, which allows approximating $[\mathbf{c}^+](\cdot)$ with fewer contractions than the method proposed in [52]. Our method also allows for higher-dimensional problems, thanks to Lohner's algorithms.

Algorithm 5 formalises our method and is illustrated in Figure 4.10. Note that:

1. *list* corresponds to a list structure, from which one can pop the first element using the function $list.pop\_front()$; and to which elements can be appended using the function $list.push\_back()$
2. *willReEnter* is a boolean variable;
3. the function *bisect*() takes a box $[\mathbf{x}]$ as an input and returns a pair of boxes $[\mathbf{x}_1]$ & $[\mathbf{x}_2]$ such that $[\mathbf{x}_1] \cup [\mathbf{x}_2] = [\mathbf{x}]$ and $[\mathbf{x}_1] \cap [\mathbf{x}_2] = \emptyset$;
4. $\overrightarrow{\mathcal{C}}_{Lohner}$ and $\overleftarrow{\mathcal{C}}_{Lohner}$ respectively correspond to the forward and backward versions of the Lohner contractor;

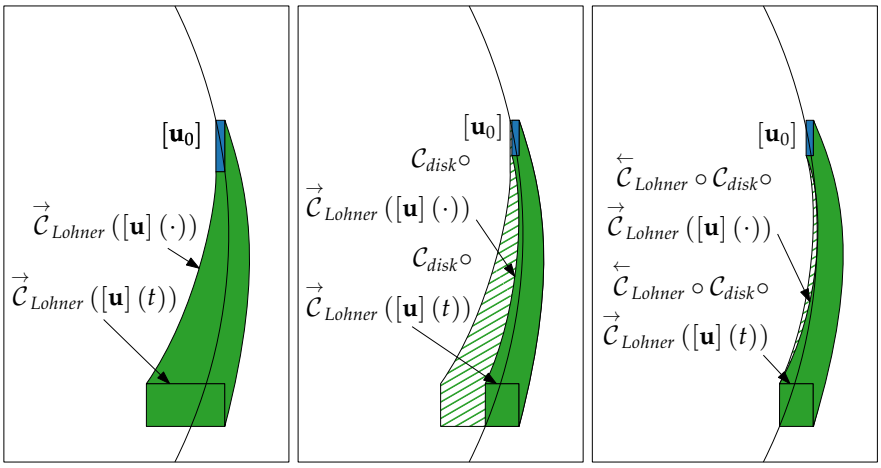


Figure 4.10: Working principle of Algorithm 5

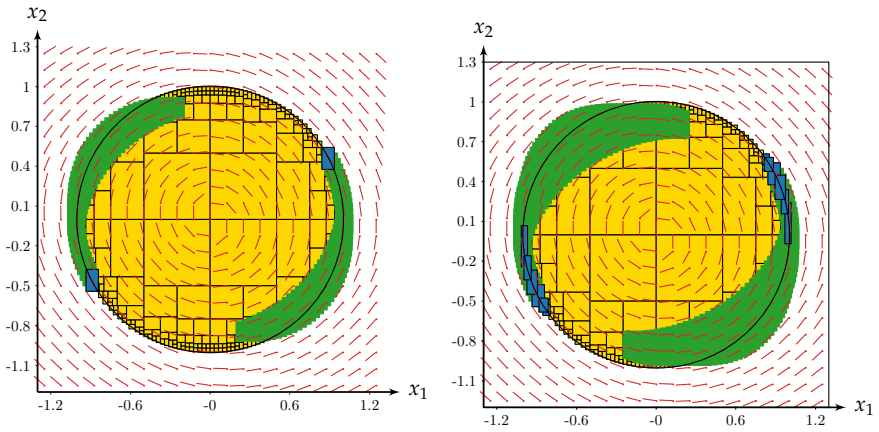
Algorithm 5 Approximating outer capture tubes using $\mathcal{C}_{Lohner}$

Input: $[\mathbf{x}]$, $[\mathbf{c}^-](\cdot)$, $N \geq 1$
$list = \{[\mathbf{x}]\}$
$[\mathbf{c}^+](\cdot) \leftarrow [\mathbf{c}^-](\cdot)$
while $list \neq \emptyset$ **do**
  $[\mathbf{u}_0] \leftarrow list.pop\_front()$
  $[\mathbf{u}_0] \leftarrow \mathcal{C}_{cross\text{-}out}([\mathbf{u}])$
  $[\mathbf{u}](t) = \begin{cases} [\mathbf{u}_0] & \text{if } t = 0; \\ \mathbb{R} & \text{otherwise.} \end{cases}$ *a*
  $willReEnter \leftarrow$ **false**
  for $i = 0$ **to** N **do**
    $[\mathbf{u}_1](\cdot) \leftarrow \overleftarrow{\mathcal{C}}_{Lohner} \circ \mathcal{C}_{disk} \circ \overrightarrow{\mathcal{C}}_{Lohner}([\mathbf{u}](\cdot))$ *b*
    if $[\mathbf{u}_1](\cdot) = \emptyset$ **then**
      $willReEnter \leftarrow$ **true**
      $\forall t \geq 0$, $[\mathbf{c}^+](t) \leftarrow [\mathbf{c}^+](t) \cup [\mathbf{u}](t)$ *c*
    end if
    $[\mathbf{u}](\cdot) \leftarrow [\mathbf{u}_1](\cdot)$
  end for
  if $willReEnter =$ **false then**
    $[\mathbf{u}_1], [\mathbf{u}_2] \leftarrow bisect([\mathbf{u}_0])$
    $list.push\_back([\mathbf{u}_1])$
    $list.push\_back([\mathbf{u}_2])$
  end if
end while

a Initialise a tube $[\mathbf{u}](\cdot)$ with the box $[\mathbf{u}_0]$
b Propagates the initial box through time, then contract the tube with respect to the interior of the disk, and propagates this new information backwards in time
c $[\mathbf{u}](\cdot)$ will re-enter the disk at some point in time and is thus part of $[\mathbf{c}^+](\cdot)$

In Figure 4.11, we represented the results obtained using our method. It is noticeable that the latter allows for fewer bisections than the approach presented in [52]. The main reason for that is that the Lohner contractor composed with $\mathcal{C}_{disk}$ allows contracting the entire tube, and therefore performs sharper guaranteed integration without additional bisections.

To sum up, our method allows finding an outer approximation of a dynamical system's capture tubes, i. e. positive invariant sets for the latter, while performing fewer bisections than the method proposed in [52]. The fact that $[\mathbf{c}^+](\cdot)$ is larger in our case than in this paper is not problematic: to improve accuracy, we could also bisect more. However, the main problem is to find such an approximation, possibly for higher-dimensional systems, and we believe that our method is more suitable for this task.



(a) The blue boxes satisfy the cross-out condition, and have re-entering trajectories (in green)

(b) Outer approximation of the capture tube (union of yellow, blue and green boxes)

Figure 4.11: Approximation of $[\mathbf{c}^+](\cdot)$ using $\mathcal{C}_{Lohner}$

A possible improvement for our method could be to take cuboids as initial boxes for the Lohner contractor, instead of boxes: doing so would reduce the wrapping effect remaining from the contraction step and therefore allow for thinner tubes. Another improvement in terms of performances could consist in using the tubes that are known to be re entrant to contract the ones that have not been contracted yet.

4.6 CONCLUSION

This paragraph concludes the presentation of the reachability analysis approach to the docking problem. The latter is based on tubes of trajectories, which allow outer approximating the set of trajectories, and thus of states, of an uncertain system. In turn, this outer approximation allows verifying two important properties:

1. that the system does not enter a forbidden area in its state space, symbolising an obstacle for example;
2. that the system can reach the desired area when initialised in a specific zone.

We successfully illustrated these properties in a simple example of a robot driven by a potential field.

Apart from the approach itself, the main contribution of this chapter is the Lohner contractor, which allows contracting a tube according to differential constraints. To the extent of our knowledge, this new contractor allows for a better contraction of the initial tube than existing contractors, if the latter is thin enough. This limitation is due to the use of Lohner's algorithm inside the contractor.

Possible improvements for the Lohner contractor could be using the CAPD library instead of our simple version of Lohner's algorithm to perform guaranteed integration, and in particular to use its automatic differentiation tool and its doubletons and tripletons sets. Another improvement that would prove extremely useful in a robotics context is the ability to deal with differential inclusions of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{u},\ \mathbf{u} \in [\mathbf{u}] \tag{4.25}$$

This could be easily implemented using the CAPD library, which proposes such a feature.

5

CONCLUSION

TABLE OF CONTENTS

5.1 CONTEXT

The offshore wind industry's fast development is pushing towards the automation of maintenance operations via underwater robots. In that context, companies started developing AUVs and autonomous ROVs capable of achieving those missions, new cheaper, lighter and more accurate sensors, and deploying new technologies allowing for faster communication. In the robotics domain, navigation and control algorithms have been adapted and improved to meet the requirements inherent to underwater operations and to allow for tasks to be performed more autonomously.

In this context, it appeared necessary to look for tools capable of validating the engineering choices made to achieve these missions. It would be inconceivable to deploy resident autonomous underwater robots in an offshore wind farm without having proven beforehand that the systems are safe to operate, i. e. displaying a behaviour validated by the executives of the project. Therefore, this thesis focus on the docking part of these maintenance missions, occurring when a robot needs to join its garage or assigned surface vessel. In particular, we chose to work on a priori validation methods to ensure feasibility of a docking mission.

5.2 CONTRIBUTIONS OF THIS THESIS

This thesis explored two different approaches to prove feasibility of a docking mission.

5.2.1 *Stability approach to the docking problem*

The first approach consists in formalising the system formed by the robot, its embedded computer, sensors and actuators, and the docking target as a stability problem. In other words, in a docking mission, the robot is expected to converge towards its target. Therefore, proving feasibility of the docking mission comes down to verifying stability of the system. The latter can either be a discrete-time, continuous-time or hybrid one, in addition to being uncertain and non-linear. Thus various methods must be used to demonstrate their stability.

Since the existing approaches did not meet all our problems' requirements, we developed new methods to prove stability of uncertain, non-linear, discrete-time, continuous-time and hybrid systems. These new methods are based on the new notion of stability contractor and the centred form of an interval function. While developing these methods, we created additional tools to help us in our task, namely algorithms to compute the centred form of a composition of interval functions iteratively.

In short, our methods allow proving asymptotic stability of a system in a chosen neighbourhood enclosing one of its fixed points. This is their main advantage compared to the existing methods for stability analysis, which cannot find such a neighbourhood but only prove its existence. An important drawback of our methods, however, is that they can only deal with small neighbourhoods. Fortunately, the latter can, in turn, be used by algorithms based on paving strategies, e. g. to approximate basins of attraction or stability regions.

In this thesis, we demonstrated how our methods could be used to prove stability of:

1. a fixed point of a discrete-time system, using the centred form as a stability contractor;
2. a periodic orbit of a discrete-time system, using the centred form as a stability contractor;
3. an equilibrium state of a continuous-time system, by discretising the system and using the centred form as a stability contractor;
4. a hybrid limit cycle of a hybrid system, by discretising the cycle using Poincaré maps and then using the centred form as a stability contractor.

Proving stability of a limit cycle of a continuous-time system could be achieved similarly than for a hybrid limit cycle.

We presented results 1 and 2 in the SNR workshop, and we submitted result 4 to the LITES journal:

- A. Bourgois and L. Jaulin, "Interval centred form for proving stability of non-linear discrete-time systems", in *SNR 2020: 6_{th} International Workshop on Symbolic-Numeric Methods for Reasoning about CPS and IoT*, Vienna, Austria, Aug. 2020. DOI: 10.4204/EPTCS.331
- A. Bourgois and L. Jaulin, "Proving the stability of a limit cycle of a hybrid system", *LITES - Leibniz Transactions on Embedded Systems*, 2020, Submitted June 2020

5.2.2 *Reachability approach to the docking problem*

The second approach developed in this thesis is based on reachability analysis. After modelling the robot and its equipment as a dynamical system, the goal was to prove that the latter could reach a specific docking area while avoiding obstacles. Since the system is uncertain and non-linear, we used tubes of trajectories to obtain mathematical proofs to reachability and non-collision.

Since we want a priori reachability proofs, our system, once modelled, needs to be simulated over time in a guaranteed way. However, the existing methods allowing to do so, namely differential constraints based tube contractors, were not performing enough, since they were not meant for long-time integration in the first place. Therefore, we developed a new tube contractor for differential constraints, based on Lohner's algorithm.

We gave a proof of concept of this approach on a simple system modelling a robot following a vector field. Since the Lohner contractor is based on Lohner's algorithm, in turn based on the centred form, only thin tubes can be contracted.

We plan on submitting a paper summarising this approach in a journal by the end of the year.

5.2.3 *Comparison of the two approaches*

We have not formally compared the performances of the two approaches presented earlier. The main goal of this thesis was indeed to develop the tools mentioned above to answer the academic requirements of this initial problem of this thesis:

- a priori proving stability of an uncertain dynamical system;
- Performing a priori reachability analysis.

Furthermore, the two methods being based on the same theoretical tool, namely the centred form, the results should not be extremely different if it comes to merely finding an initial condition such that the system will dock onto its target.

However, both approaches can be compared in terms of feature: the first one allows proving stability of a dynamical system, and this property could be extended to problems utterly different than the docking one; and the second one is based on a temporal contractor for tubes, which can also be used in other contexts, such as underwater localisation.

5.2.4 *Other contributions*

This thesis also aimed at introducing various existing tools related to dynamical systems. Therefore, we introduced the CAPD library via a few code examples all along this document, while giving an overview of the related mathematical notions. We also gave an extensive introduction to Lohner's algorithm, which we believe could be used in more robotics-related applications, despite its limitation to small boxes.

5.3 PROSPECTS

The work started in this thesis could be continued in multiple ways. First, some unpublished results, among which Lohner's contractors and its applications, and proving stability of continuous-time systems, remain to be published. We also plan to release a clean version of the software developed in this thesis to ease interactions between CAPD, Tubex and Ibex, as well as the implementations of the different algorithms presented in this thesis.

Secondly, we have seen that the centred form algorithm we presented in Chapter 3 could be used as a stability contractor. It can also be seen as a guaranteed integration algorithm, although less performing than Lohner's algorithm or the ones implemented in CAPD. An approach allowing to obtain larger stability neighbourhoods for the studied system could be to verify whether these algorithms can also be used as stability contractors.

Thirdly, the Lohner contractor we presented in Chapter 4 could be used differently than in this thesis. Indeed, while our approach required a priori proofs, this contractor could be used during or after

a mission to contract the tube of trajectories of the robot, in a context of loop detection in the trajectories for example. To make sure the uncertainties remain small, ensuring the contractor's proper functioning, a composition with other contractors, e. g. static ones dealing with measurement data, could be of help.

Additionally, even if the simple implementation of Lohner's algorithm we presented in Chapter 2 already enables better performances of the Lohner contractor than its competitors, results could be improved by using the CAPD library instead of our implementation. This would allow us to enjoy the optimisations made inside this library in the contractor (automatic differentiation, sharper representable sets, automatic computation of integration step...).

The reader might recall Remark 2.11, that pointed out that we could only deal with differential inclusion having constant parameters in this thesis. However, methods exist to integrate general differential inclusion in a guaranteed manner, some of which are implemented in CAPD. General differential inclusions (i. e. with non-constant parameters) can be of use to model uncertain systems. It would be interesting to improve the Lohner contractor to deal with general differential inclusions. This would allow for broader use of the contractor.

Finally, this work mainly consists in theoretical tools at the moment. We have not applied them yet to real-life systems. This is a longer-term goal, as extra work would be required to formalise these real-life systems and characterise their parameters.

BIBLIOGRAPHY

[1] R. Alur, "Formal verification of hybrid systems", in *Proceedings of the ninth ACM international conference on Embedded software*, 2011, pp. 273–278. DOI: 10.1145/2038642.2038685.

[2] E. Asarin, T. Dang, and A. Girard, "Hybridization methods for the analysis of nonlinear systems", *Acta Informatica*, vol. 43, no. 7, pp. 451–476, 2007. DOI: 10.1007/s00236-006-0035-7.

[3] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems", in *International Conference on Computer Aided Verification*, Springer, 2002, pp. 365–370. DOI: 10.1007/3-540-45657-0_30.

[4] E. Asarin, O. Maler, and A. Pnueli, "Reachability analysis of dynamical systems having piecewise-constant derivatives", *Theoretical computer science*, vol. 138, no. 1, pp. 35–65, 1995. DOI: 10.1016/0304-3975(94)00228-B.

[5] J.-P. Aubin and H. Frankowska, *Set-valued analysis*. Birkhäuser, Basel, 2009. DOI: 10.1007/978-0-8176-4848-0.

[6] E. Auer, S. Kiel, and A. Rauh, "A verified method for solving piecewise smooth initial value problems", *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 4, pp. 731–747, 2013. DOI: 10.2478/amcs-2013-0055.

[7] C. Bell, M. Bayliss, and R. Warburton, *Handbook for ROV pilot/technicians*, 2nd edition. Oilfield Publ. Limited, 2002.

[8] A. Bethencourt and L. Jaulin, "Solving non-linear constraint satisfaction problems involving time-dependant functions", *Mathematics in Computer Science*, vol. 8, no. 3-4, pp. 503–523, 2014. DOI: 10.1007/s11786-014-0209-6.

[9] F. Blanchini and S. Miani, *Set-theoretic methods in control*. Birkhäuser, Cham, 2008. DOI: 10.1007/978-0-8176-4606-6.

[10] A. Bourgois and L. Jaulin, "Interval centred form for proving stability of non-linear discrete-time systems", in *SNR 2020: 6_{th} International Workshop on Symbolic-Numeric Methods for Reasoning about CPS and IoT*, Vienna, Austria, Aug. 2020. DOI: 10.4204/EPTCS.331.

[11] ——, "Proving the stability of a limit cycle of a hybrid system", *LITES - Leibniz Transactions on Embedded Systems*, 2020, Submitted June 2020.

[12] T. Bourke and M. Pouzet, "Zélus: A synchronous language with ODEs", in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, 2013, pp. 113–118. DOI: 10.1145/2461328.2461348.

[13] F. Boyer, V. Lebastard, C. Chevallereau, S. Mintchev, and C. Stefanini, "Underwater navigation based on passive electric sense: new perspectives for underwater docking", *The International Journal of Robotics Research*, vol. 34, no. 9, pp. 1228–1250, 2015. DOI: 10.1177/0278364915572071.

[14] M. S. Branicky, "Stability of hybrid systems: state of the art", in *Proceedings of the 36th IEEE Conference on Decision and Control*, vol. 1, 1997, pp. 120–125. DOI: 10.1109/CDC.1997.650600.

[15] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: model and optimal control theory", *IEEE transactions on automatic control*, vol. 43, no. 1, pp. 31–45, 1998. DOI: 10.1109/9.654885.

[16] C. Carbonnel, G. Trombettoni, P. Vismara, and G. Chabert, "Q-intersection algorithms for constraint-based robust parameter estimation", in *AAAI Conference on Artificial Intelligence*, 2014, pp. 2630–2636.

[17] G. Chabert *et al.*, *The ibex library*, http://www.ibex-lib.org/, 2007.

[18] G. Chabert and L. Jaulin, "Contractor Programming", *Artificial Intelligence*, vol. 173, pp. 1079–1100, 2009. DOI: 10.1016/j.artint.2009.03.002.

[19] R. D. Christ and R. L. Wernli Sr., *The ROV manual: a user guide for remotely operated vehicles*. Butterworth-Heinemann, 2013.

[20] J.-L. Colaço, B. Pagano, and M. Pouzet, "Scade 6: A formal language for embedded critical software development", in *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, IEEE, 2017, pp. 1–11. DOI: 10.1109/TASE.2017.8285623.

[21] P. Collins and A. Goldsztejn, "The reach-and-evolve algorithm for reachability analysis of nonlinear dynamical systems", *Electronic Notes in Theoretical Computer Science*, vol. 223, pp. 87–102, 2008. DOI: 10.1016/j.entcs.2008.12.033.

[22] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Springer, 2017, vol. 118. DOI: 10.1007/978-3-642-20144-8.

[23] G. F. Corliss and R. Rihm, "Validating an a priori enclosure using high-order Taylor series", *Mathematical Research*, vol. 90, pp. 228–238, 1996.

[24] S. Cowen, S. Briest, and J. Dombrowski, "Underwater docking of autonomous undersea vehicles using optical terminal guidance", in *Oceans' 97. MTS/IEEE Conference Proceedings*, IEEE, vol. 2, 1997, pp. 1143–1147. DOI: 10.1109/OCEANS.1997.624153.

[25] V. Creuze, "Robots marins et sous-marins - perception, modélisation, commande", *Techniques de l'ingénieur Applications en robotique*, 2014. [Online]. Available: https://www.techniques-ingenieur.fr/base-documentaire/automatique-robotique-th16/applications-en-robotique-42623210/robots-marins-et-sous-marins-s7783/.

[26] N. Delanoue, L. Jaulin, and B. Cottenceau, "An algorithm for computing a neighborhood included in the attraction domain of an asymptotically stable point", *Communications in Nonlinear Science and Numerical Simulation*, vol. 21, no. 1-3, pp. 181–189, 2015. DOI: 10.1016/j.cnsns.2014.08.034.

[27] J. A. dit Sandretto and A. Chapoutot, "Validated explicit and implicit Runge–Kutta methods", *Reliable Computing*, vol. 22, no. 1, pp. 79–103, Jul. 2016.

[28] P. Eijgenraam, "The solution of initial value problems using interval arithmetic: formulation and analysis of an algorithm", *MC Tracts*, 1981.

[29] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: large-scale direct monocular SLAM", in *Computer Vision – ECCV 2014*, Springer International Publishing, 2014, pp. 834–849. DOI: 10.1007/978-3-319-10605-2_54.

[30] J. Evans, P. Redmond, C. Plakas, K. Hamilton, and D. Lane, "Autonomous docking for Intervention-AUVs using sonar and video-based real-time 3D pose estimation", in *Oceans' 2003. Celebrating the Past... Teaming Toward the Future (IEEE Cat. No. 03CH37492)*, IEEE, vol. 4, 2003, pp. 2201–2210. DOI: 10.1109/OCEANS.2003.178243.

[31] M. D. Feezor, F. Y. Sorrell, P. R. Blankinship, and J. G. Bellingham, "Autonomous underwater vehicle homing/docking via electromagnetic guidance", *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 515–521, 2001. DOI: 10.1109/48.972086.

[32] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011. DOI: 10.1002/9781119994138.

[33] T. Fukasawa, T. Noguchi, T. Kawasaki, and M. Baino, ""MARINE BIRD", a new experimental AUV with underwater docking and recharging system", in *Oceans 2003. Celebrating the Past... Teaming Toward the Future (IEEE Cat. No. 03CH37492)*, IEEE, vol. 4, 2003, pp. 2195–2200. DOI: 10.1109/OCEANS.2003.178242.

[34] A. Girard, "Computation and stability analysis of limit cycles in piecewise linear hybrid systems", *IFAC Proceedings Volumes*, vol. 36, no. 6, pp. 181–186, 2003. DOI: 10.1016/S1474-6670(17)36428-5.

[35] ——, "Analyse algorithmique des systèmes hybrides", PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2004.

[36] ——, "Reachability of uncertain linear systems using zonotopes", in *Hybrid Systems: Computation and Control*, Springer Berlin Heidelberg, 2005, pp. 291–305. DOI: 10.1007/978-3-540-31954-2_19.

[37] M. Giunti, *Computation, dynamics, and cognition*. Oxford University Press, 1997.

[38] E. Goubault, O. Mullier, S. Putot, and M. Kieffer, "Inner approximated reachability analysis", in *Proceedings of the 17th international conference on Hybrid systems: computation and control*, 2014, pp. 163–172. DOI: 10.1145/2562059.2562113.

[39] J. Hajjami, J. Caracotte, G. Caron, and T. Napoleon, "ArUcOmni: detection of highly reliable fiducial markers in panoramic images", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 634–635. DOI: 10.1109/CVPRW50498.2020.00325.

[40] S. S. Haykin, *Neural networks and learning machines*, Third. Upper Saddle River, NJ: Pearson Education, 2009.

[41] M. W. Hirsch, S. Smale, and R. L. Devaney, *Differential equations, dynamical systems, and an introduction to chaos*. Academic press, 2013. DOI: 10.1016/C2009-0-61160-0.

[42] I. A. Hiskens, "Stability of limit cycles in hybrid systems", in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, IEEE, 2001, 6–pp. DOI: 10.1109/HICSS.2001.926280.

[43] I. A. Hiskens and M. A. Pai, "Trajectory sensitivity analysis of hybrid systems", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 2, pp. 204–220, 2000. DOI: 10.1109/81.828574.

[44] IEA, "Key World Energy Statistics", International Energy Agency, Paris, Tech. Rep., Aug. 2020, https://www.iea.org/reports/key-world-energy-statistics-2020.

[45] F. Immler and C. Traut, "The flow of ODEs: formalization of variational equation and Poincaré map", *Journal of Automated Reasoning*, vol. 62, no. 2, pp. 215–236, Feb. 2019, ISSN: 1573-0670. DOI: 10.1007/s10817-018-9449-5.

[46] L. W. Jackson, "Interval arithmetic error-bounding algorithms", *SIAM Journal on Numerical Analysis*, vol. 12, no. 2, pp. 223–238, 1975. DOI: 10.1137/0712021.

[47] T. Jackson and A. Radunskaya, *Applications of dynamical systems in biology and medicine*. Springer, 2015, vol. 158. DOI: 10.1007/978-1-4939-2782-1.

[48] L. Jaulin, *Mobile robotics*. John Wiley & Sons, 2019. DOI: 10.1002/9781119663546.

[49] L. Jaulin and J. Burger, "Proving stability of uncertain parametric models", *Automatica*, p. 627, 1999.

[50] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. London: Springer-Verlag, 2001. DOI: 10.1007/978-1-4471-0249-6.

[51] L. Jaulin and F. Le Bars, "An interval approach for stability analysis: Application to sailboat robotics", *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 282–287, 2012. DOI: 10.1109/TRO.2012.2217794.

[52] L. Jaulin, D. Lopez, V. Le Doze, S. Le Menec, J. Ninin, G. Chabert, M. S. Ibnseddik, and A. Stancu, "Computing capture tubes", in *International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*, Springer, 2015, pp. 209–224. DOI: 10.1007/978-3-319-31769-4_17.

[53] L. Jaulin and E. Walter, "Set inversion via interval analysis for nonlinear bounded-error estimation", *Automatica*, vol. 29, no. 4, pp. 1053–1064, 1993. DOI: 10.1016/0005-1098(93)90106-4.

[54] K. H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, "On the regularization of Zeno hybrid automata", *Systems & control letters*, vol. 38, no. 3, pp. 141–150, 1999. DOI: 10.1016/S0167-6911(99)00059-6.

[55] H. Joudrier, "Guaranteed deterministic global optimization using constraint programming through algebraic, functional and piecewise differential constraints", PhD thesis, Université Grenoble Alpes, 2018.

[56] T. Kapela, M. Mrozek, D. Wilczak, and P. Zgliczyński, "CAPD:: DynSys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems", *Communications in Nonlinear Science and Numerical Simulation*, in press 2020. DOI: 10.1016/j.cnsns.2020.105578.

[57] T. Kapela and P. Zgliczyński, "The existence of simple choreographies for the N-body problem – a computer-assisted proof", *Nonlinearity*, vol. 16, no. 6, p. 1899, 2003. DOI: 10.1088/0951-7715/16/6/302.

[58] N. Kato and M. Endo, "Guidance and control of unmanned, untethered submersible for rendezvous and docking with underwater station", in *Proceedings OCEANS*, IEEE, vol. 3, 1989, pp. 804–809. DOI: 10.1109/OCEANS.1989.586685.

[59] H. K. Khalil and J. W. Grizzle, *Nonlinear systems*. Prentice hall Upper Saddle River, NJ, 2002, vol. 3.

[60] F. Krückeberg, "Ordinary differential equations", in *Topics in interval analysis*. Clarendon Press Oxford, 1969.

[61] S. Krupinski, F. Maurelli, G. Grenon, and Y. Petillot, "Investigation of autonomous docking strategies for robotic operation on intervention panels", in *Oceans' 2008*, IEEE, 2008, pp. 1–10. DOI: 10.1109/OCEANS.2008.5151995.

[62] W. Kühn, "Rigorously computed orbits of dynamical systems without the wrapping effect", *Computing*, vol. 61, no. 1, pp. 47–67, 1998. DOI: 10.1007/BF02684450.

[63] 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*, Springer, 1993, pp. 122–188. DOI: 10.1007/978-1-4612-0349-0_4.

[64] J. C. J. Lambiotte, R. Coulson, S. M. Smith, and E. An, "Results from mechanical docking tests of a morpheus class AUV with a dock designed for an OEX class AUV", in *Oceans'02 MTS/IEEE*, IEEE, vol. 1, 2002, pp. 260–265. DOI: 10.1109/OCEANS.2002.1193281.

[65] M. Laranjeira, C. Dune, and V. Hugel, "Catenary-based visual servoing for tethered robots", in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 732–738. DOI: 10.1109/ICRA.2017.7989090.

[66] ——, "Catenary-based visual servoing for tether shape control between underwater vehicles", *Ocean Engineering*, vol. 200, p. 107 018, 2020. DOI: 10.1016/j.oceaneng.2020.107018.

[67] F. Le Bars, J. Sliwka, L. Jaulin, and O. Reynet, "Set-membership state estimation with fleeting data", *Automatica*, vol. 48, no. 2, pp. 381–387, 2012. DOI: 10.1016/j.automatica.2011.11.004.

[68] T. Le Mézo, L. Jaulin, and B. Zerr, "An interval approach to compute invariant sets", *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 4236–4242, 2017. DOI: 10.1109/TAC.2017.2685241.

[69] ——, "Bracketing backward reach sets of a dynamical system", *International Journal of Control*, pp. 1–13, 2019. DOI: 10.1080/00207179.2019.1643910.

[70] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[71] M. Lhommeau, L. Jaulin, and L. Hardouin, "Inner and outer approximation of capture basins using interval analysis", in *4th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2007.

[72] A. Liapounoff, "Problème général de la stabilité du mouvement", in *Annales de la Faculté des sciences de Toulouse: Mathématiques*, vol. 9, 1907, pp. 203–474.

[73] R. J. Lohner, "Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems", in *Institute of mathematics and its applications conference series*, Oxford University Press, vol. 39, 1992, pp. 425–425.

[74] ——, "On the ubiquity of the wrapping effect in the computation of error bounds", in *Perspectives on enclosure methods*, Springer, 2001, pp. 201–216. DOI: 10.1007/978-3-7091-6282-8_12.

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

[76] E. N. Lorenz, "Deterministic nonperiodic flow", *Journal of the atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[77] A. Martins, J. M. Almeida, H. Ferreira, H. Silva, N. Dias, A. Dias, C. Almeida, and E. P. Silva, "Autonomous surface vehicle docking manoeuvre with visual information", in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 4994–4999. DOI: 10.1109/ROBOT.2007.364249.

[78] N. Metropolis and S. Ulam, "The Monte Carlo method", *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[79] D. Monnet, L. Jaulin, J. Ninin, A. Chapoutot, and J. Alexandre dit Sandretto, "Inner and Outer Computation of the Viability Kernel based on Interval Analysis", in *International Symposium on Set Membership - Applications, Reliability and Theory*, Manchester, United Kingdom, Sep. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01371498.

[80] ——, "Viability kernel computation based on interval methods", in *Small Workshop on Interval Methods*, Prague, Czech Republic, Jun. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01371484.

[81] R. E. Moore, "The automatic analysis and control of error in digital computation based on the use of interval numbers", in *Error in Digital Computation, Vol. 1*, L. B. Rall, Ed., John Wiley & Sons, 1965.

[82] ——, *Interval analysis*. Prentice-Hall Englewood Cliffs, 1966, vol. 4.

[83] ——, *Methods and applications of interval analysis*. SIAM, 1979. DOI: 10.1137/1.9781611970906.

[84] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. SIAM, 2009. DOI: 10.1137/1.9780898717716.

[85] M. Mrozek and P. Zgliczyński, "Set arithmetic and the enclosing problem in dynamics", in *Annales Polonici Mathematici*, Instytut Matematyczny Polskiej Akademii Nauk, vol. 74, 2000, pp. 237–259. DOI: 10.4064/ap-74-1-237-259.

[86] H. E. Murdock, D. Gibb, and T. André, "Renewables 2020 Global Status Report", Renewable Energy Policy Network for the 21st Century, Paris: REN21 Secretariat, Tech. Rep., 2020, https://www.ren21.net/reports/global-status-report/.

[87] N. S. Nedialkov and K. R. Jackson, "An interval Hermite-Obreschkoff method for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation", *Reliable Computing*, vol. 5, no. 3, pp. 289–310, 1999. DOI: 10.1007/978-94-017-1247-7_23.

[88] ——, "A new perspective on the wrapping effect in interval methods for initial value problems for ordinary differential equations", in *Perspectives on Enclosure Methods*, Springer, 2001, pp. 219–263. DOI: 10.1007/978-3-7091-6282-8_13.

[89] N. S. Nedialkov, K. R. Jackson, and J. D. Pryce, "An effective high-order interval method for validating existence and uniqueness of the solution of an IVP for an ODE", *Reliable Computing*, vol. 7, no. 6, pp. 449–465, 2001. DOI: 10.1023/A:1014798618404.

[90] B. Neveu, M. De La Gorce, and G. Trombettoni, "Improving a constraint programming approach for parameter estimation", in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2015, pp. 852–859. DOI: 10.1109/ICTAI.2015.164.

[91] N. Palomeras, P. Ridao, D. Ribas, and G. Vallicrosa, "Autonomous I-AUV docking for fixed-base manipulation", *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 12 160–12 165, 2014. DOI: 10.3182/20140824-6-ZA-1003.01878.

[92] L. Perko, *Differential equations and dynamical systems*. Springer Science & Business Media, 2013, vol. 7. DOI: 10.1007/978-1-4613-0003-8.

[93] G. J. S. Rae and S. M. Smith, "A fuzzy rule based docking procedure for autonomous underwater vehicles", in *Oceans' 92. Proceedings: Mastering the Oceans Through Technology*, IEEE, vol. 2, 1992, pp. 539–546. DOI: 10.1109/OCEANS.1992.607638.

[94] L. B. Rall, "Automatic Differentiation - Technique and Applications", *Lecture Notes in Computer Science*, vol. 120, 1981.

[95] N. Ramdani and N. S. Nedialkov, "Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques", *Nonlinear Analysis: Hybrid Systems*, vol. 5, no. 2, pp. 149–162, 2011. DOI: 10.1016/j.nahs.2010.05.010.

[96] S. Ratschan and Z. She, "Providing a basin of attraction to a target region of polynomial systems by computation of Lyapunov-like functions", *SIAM Journal on Control and Optimization*, vol. 48, no. 7, pp. 4377–4394, 2010. DOI: 10.1137/090749955.

[97] A. Rauh, J. Kersten, and H. Aschemann, "Techniques for verified reachability analysis of quasi-linear continuous-time systems", in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2019, pp. 18–23. DOI: 10.1109/MMAR.2019.8864648.

[98] A. Rauh, J. Kersten, and H. Aschemann, "Interval methods and contractor-based branch-and-bound procedures for verified parameter identification of quasi-linear cooperative system models", *Journal of Computational and Applied Mathematics*, vol. 367, 2020. DOI: 10.1016/j.cam.2019.112484.

[99] A. Rauh, M. Kletting, H. Aschemann, and E. P. Hofer, "Interval methods for simulation of dynamical systems with state-dependent switching characteristics", in *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, IEEE, 2006, pp. 355–360. DOI: 10.1109/CACSD-CCA-ISIC.2006.4776672.

[100] R. Rihm, "Enclosing solutions with switching points in ordinary differential equations", *Computer arithmetic and enclosure methods. Proceedings of SCAN*, vol. 91, pp. 419–425, 1992.

[101] J. Rohn, "An algorithm for checking stability of symmetric interval matrices", *IEEE Transactions on Automatic Control*, vol. 41, no. 1, pp. 133–136, 1996. DOI: 10.1109/9.481618.

[102] ——, "Bounds on eigenvalues of interval matrices", *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 78, no. S3, pp. 1049–1050, 1998. DOI: 10.1002/zamm.19980781593.

[103] S. Rohou *et al.*, *The tubex library – constraint-programming for robotics*, http://simon-rohou.fr/research/tubex-lib/, 2017.

[104] S. Rohou, L. Jaulin, L. Mihaylova, F. Le Bars, and S. M. Veres, "Guaranteed computation of robot trajectories", *Robotics and Autonomous Systems*, vol. 93, pp. 76–84, 2017. DOI: 10.1016/j.robot.2017.03.020.

[105] ——, *Reliable robot localization: a constraint-programming approach over dynamical systems*. John Wiley & Sons, 2019. DOI: 10.1002/9781119680970.

[106] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers", *Image and vision Computing*, vol. 76, pp. 38–47, 2018. DOI: 10.1016/j.imavis.2018.05.004.

[107] S. M. Rump, "Algebraic computation, numerical computation and verified inclusions", in *Trends in computer algebra*, Springer, 1988, pp. 177–197. DOI: 10.1007/3-540-18928-9_13.

[108] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011. DOI: 10.1109/ICRA.2011.5980567.

[109] P. Saint-Pierre, "Hybrid kernels and capture basins for impulse constrained systems", in *International Workshop on Hybrid Systems: Computation and Control*, Springer, 2002, pp. 378–392. DOI: 10.1007/3-540-45873-5_30.

[110] E. Schrödinger and R. Penrose, *'Nature and the Greeks' and 'Science and Humanism'*, ser. Canto original series. Cambridge University Press, 1996. DOI: 10.1017/CBO9781139878333.

[111] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016. DOI: 10.1007/978-3-319-32552-1.

[112] S. N. Simić, K. H. Johansson, S. Sastry, and J. Lygeros, "Towards a geometric theory of hybrid systems", in *International Workshop on Hybrid Systems: Computation and Control*, Springer, 2000, pp. 421–436. DOI: 10.1007/3-540-46430-1_35.

[113] S. N. Simić, S. Sastry, K. H. Johansson, and J. Lygeros, "Hybrid limit cycles and hybrid Poincaré-Bendixson", *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 197–202, 2002. DOI: 10.3182/20020721-6-ES-1901.01104.

[114] H. Singh, J. G. Bellingham, F. Hover, S. Lemer, B. A. Moran, K. Von der Heydt, and D. Yoerger, "Docking for an autonomous ocean sampling network", *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 498–514, 2001. DOI: 10.1109/48.972084.

[115] S. M. Smith, G. J. S. Rae, and D. T. Anderson, "Applications of fuzzy logic to the control of an autonomous underwater vehicle", in *[Proceedings 1993] Second IEEE International Conference on Fuzzy Systems*, IEEE, 1993, pp. 1099–1106. DOI: 10.1109/FUZZY.1993.327361.

[116] E. D. Sontag, *Mathematical control theory: deterministic finite dimensional systems*. Springer Science & Business Media, 2013, vol. 6. DOI: 10.1007/978-1-4612-0577-7.

[117] R. Stokey, M. Purcell, N. Forrester, T. Austin, R. Goldsborough, B. Allen, and C. von Alt, "A docking system for REMUS, an autonomous underwater vehicle", in *Oceans' 97. MTS/IEEE Conference Proceedings*, IEEE, vol. 2, 1997, pp. 1132–1136. DOI: 10.1109/OCEANS.1997.624151.

[118] W. Taha, A. Duracz, Y. Zeng, K. Atkinson, F. A. Bartha, P. Brauner, J. Duracz, F. Xu, R. Cartwright, M. Konečný, *et al.*, "Acumen: An open-source testbed for cyber-physical systems research", in *International Internet of Things Summit*, Springer, 2015, pp. 118–130. DOI: 10.1007/978-3-319-47063-4_11.

[119] G. Teschl, *Ordinary differential equations and dynamical systems*. American Mathematical Soc., 2012, vol. 140. DOI: 10.1090/gsm/140.

[120] The CAPD group, *CAPD - Computer Assisted Proofs in Dynamics, a package for rigorous numerics*, http://capd.ii.uj.edu.pl, 2005.

[121] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. Cambridge, Massachusetts: The MIT Press, 2006. DOI: 10.1145/504729.504754.

[122] P. Trslic, M. Rossi, L. Robinson, C. W. O'Donnel, A. Weir, J. Coleman, J. Riordan, E. Omerdic, G. Dooly, and D. Toal, "Vision based autonomous docking for work class ROVs", *Ocean Engineering*, vol. 196, p. 106 840, 2020. DOI: 10.1016/j.oceaneng.2019.106840.

[123] W. Tucker, "The Lorenz attractor exists", *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, vol. 328, no. 12, pp. 1197–1202, 1999. DOI: 10.1016/S0764-4442(99)80439-X.

[124] ——, "A Rigorous ODE Solver and Smale's 14th Problem", *Foundations of Computational Mathematics*, vol. 2, no. 1, pp. 53–117, 2002. DOI: 10.1007/s002080010018.

[125] ——, "Computing accurate Poincaré maps", *Physica D: Nonlinear Phenomena*, vol. 171, no. 3, pp. 127–137, 2002. DOI: 10.1016/S0167-2789(02)00603-6.

[126] G. Vallicrosa, J. Bosch, N. Palomeras, P. Ridao, M. Carreras, and N. Gracias, "Autonomous homing and docking for AUVs using range-only localization and light beacons", *IFAC-papersonline*, vol. 49, no. 23, pp. 54–60, 2016. DOI: 10.1016/j.ifacol.2016.10.321.

[127] B. Van der Pol, "LXXXVIII. On "relaxation-oscillations"", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 978–992, 1926.

[128] L. Von Stumberg, V. Usenko, and D. Cremers, "Direct sparse visual-inertial odometry using dynamic marginalization", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2510–2517. DOI: 10.1109/ICRA.2018.8462905.

[129] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 4193–4198. DOI: 10.1109/IROS.2016.7759617.

[130] R. Wang, M. Schworer, and D. Cremers, "Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras", in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3903–3911. DOI: 10.1109/ICCV.2017.421.

[131] D. Wilczak, *Computer-assisted proofs in dynamics Part I: Topological methods*, Available at https://ww2.ii.uj.edu.pl/~wilczak/capd-tutorial/CAPD_tutorial_part_I.pdf, Tutorial presented at the 8th Small Workshop on Interval Methods (SWIM), 2016.

[132] D. Wilczak and P. Zgliczyński, "Heteroclinic connections between periodic orbits in planar restricted circular three-body problem – a computer assisted proof", *Communications in mathematical physics*, vol. 234, no. 1, pp. 37–75, 2003. DOI: 10.1007/s00220-002-0709-0.

[133] ——, "Heteroclinic connections between periodic orbits in planar restricted circular three body problem. Part II", *Communications in mathematical physics*, vol. 259, no. 3, pp. 561–576, 2005. DOI: 10.1007/s00220-005-1374-x.

[134] ——, "C^r-Lohner algorithm", *Schedae Informaticae*, vol. 20, pp. 9–46, 2011. DOI: 10.4467/20838476SI.11.001.0287.

[135] P. Zgliczyński, "C^1-lohner algorithm", *Foundations of Computational Mathematics*, vol. 2, no. 4, pp. 429–465, 2002. DOI: 10.1007/s102080010025.

COLOPHON

This thesis has been typeset in LaTeX, using the great `classicthesis`[1] template, developed by André Miede.

Final Version as of February 22, 2021.

1 https://www.miede.de/

DOCTORAT BRETAGNE LOIRE / MATHSTIC

ENSTA Bretagne

FORSSEA ROBOTICS

Lab-STICC

Titre : Amarrage collaboratif automatique et sécurisé d'un robot sur une plateforme mobile

Mots-clés : docking sous-marin, stabilité des systèmes dynamiques, systèmes hybrides, programmation par contraintes, intégration garantie

Résumé : La multiplication des installations offshore suscite un besoin de robots autonomes fiables, capable d'effectuer des missions d'inspection et de maintenance tout en minimisant les coûts opérationnels. Pour réduire le risque d'accident pendant une mission, des outils mathématiques peuvent être utilisés pour démontrer a priori son bon déroulement. Dans cette thèse, des nouvelles méthodes reposant sur une approche ensembliste sont présentées à cet effet.

Premièrement, nous proposons une nouvelle méthode pour analyser la stabilité d'un système incertain discret, continue ou hybride. Ensuite, nous présentons une approche s'inspirant de l'analyse d'atteignabilité, pour laquelle nous avons développé un nouvel outil de programmation par contraintes permettant d'implémenter des contraintes différentielles. Ces deux approches permettent de prédire le comportement d'un robot avant même son déploiement.

Ces outils sont illustrés par des exemples réalistes issus des domaines de la localisation et du contrôle, appliqués au problème d'amarrage sous-marin. De plus, nous présentons la librairie CAPD dans un contexte robotique grâce à des exemples pratiques.

Title: Safe & collaborative autonomous underwater docking

Keywords: underwater docking, stability of dynamical systems, hybrid systems, constraint programming, guaranteed integration

Abstract: The increasing number of offshore facilities triggers the need for reliable autonomous robots to perform inspection and maintenance missions, while minimising operational expenses. In order to decrease the likelihood of undesired events during a mission, mathematical tools can be used to prove a priori its feasibility. In the following, new methods based on a set-membership approach are developed and presented in this regard.

First, we propose a new method to analyse stability of a discrete, continuous and hybrid uncertain system. Alternatively, we present an approach based on reachability analysis for which we developed a novel constraint programming tool to implement differential constraints. Both approaches allow predicting the behaviour of a robot before its actual deployment.

These tools are illustrated with realistic examples falling in the field of localisation & control, and in particular applied to underwater docking. Furthermore, the Computer-Assisted Proofs in Dynamics (CAPD) library is introduced in a robotics context via practical examples.