THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : STIC / Automatique
École Doctorale SICMA

présentée par
Aymeric Bethencourt
Préparée à l'ENSTA Bretagne (ex ENSIETA), Equipe OSM, Pôle STIC

Analyse par intervalles pour la localisation en essaim. Application à la robotique sous-marine.

Thèse soutenue le 30 septembre 2014
devant le jury composé de :

Luc JAULIN, directeur de thèse
Professeur, Université de Bretagne Occidentale, LAB-STICC, Brest.

Philippe BONNIFAIT, rapporteur
Professeur, Université de Technologie de Compiègne, Lab Heudiasyc, Compiègne.

Simon LACROIX, rapporteur
Professeur, CNRS, Laboratory for Analysis and Architecture of Systems, Toulouse.

Laurent HARDOUIN, examinateur
Professeur, Université d'Angers, Angers.

Benoit ZERR, examinateur
Professeur, Université de Bretagne Occidentale, LAB-STICC, Brest.

Gilles CHABERT, examinateur
Maître de conférences, École des Mines de Nantes, Nantes.

Gaël DESILLES, invitée
Responsable métier guidage-navigation, DGA, Bagneux.

ENSTA Bretagne

DGA

Prey is a novel by *Michael Crichton*, first published in November 2002. The novel tells the story of researchers that created a swarm of millions of robots in the nanometer range. The goal of this swarm was to be an autonomous camera for military applications.

However, it escaped the control of the researchers and became self-aware and self-replicating, posing a threat for all human beings. This apocalyptic scenario among many others shows the fear for scientific advances and technological progress.

What if we lose control?

Interval Analysis for swarm localization. Application to underwater robotics.

Aymeric BETHENCOURT

September 29, 2014

Contents

List of Figures

Thanks

I would like to thank :

- Luc JAULIN for his guidance as my supervisor,

- Laurent HARDOUIN for following my progresses over the years,

- Gilles CHABERT for his assistance with IBEX,

- Tamara BRIZARD for her support and for spellchecking this thesis,

- Fabrice LE BARS for his help on many technical issues,

- Annick BILLON COAT for her help with administrative paperwork,

- Simon LACROIX and Philippe BONNIFAIT for being my reporters and their precious insights,

- and finally the persons at the DGA that gave me the opportunity to realize this Ph.D.

I would not have been able to do it without you.

Chapter 1

Introduction

1.1 Context

Localization and spatial awareness are fundamental to any application involving mobility, and essential to life on earth. The human body needs to be localized in space to move towards a goal. The main sensors used by humans for localization are the eyes and the vestibular system, dedicated to balance. Just like the human body, a good localization is essential for any mobile robotics application. However robots do not yet have the same computing capability as the human brain. Although it is possible to use sensors similar to the eyes; like cameras, or similar to the vestibular system, like gyroscopes, robots cannot yet efficiently localize themselves based solemnly on these sensors. Robots often use the *Global Positioning System (GPS)*, a space-based satellite navigation system that provides location and time information anywhere on or near the Earth. However, what happens when the GPS is not; or partially not, available? For many indoor or underwater applications, the GPS is simply not a solution as the high frequency electromagnetic waves that it uses barely propagate in these environments.

As the number of underwater operations increases every year, the need for autonomous underwater robots becomes greater. My recent interview on the subject for national television is a testament to the flourishing interest on the subject from both the public and private sectors (interview available on `http://youtu.be/Zwjbufay9Z0`). As the interviewer states, renewable marine energies and the exploitation of underwater resources are fields that are expanding exponentially. These resources can be exploited through deep-water off-shore structures that require regular inspection. This job is usually done by operators onboard submarines, which can be perilous, or remotely by operators using *Remotely Operated Vehicles (ROVs)*. The latter has multiple drawbacks including a heavy cable (called umbilical cable) needed to connect the station to the ROV for control and power. Regardless of the method used, the work stays repetitive and laborious for the operator. However; this work can be automated using *Automated Underwater Vehicles (AUVs)* [Veres et al., 2008], which do not need an umbilical cable to operate. Neither do they need an operator as the mission is done autonomously using the embedded algorithms in the AUV. To accomplish this, they need the best possible localization to reach their objective and return to the station [Baccou and Jouvencel, 2002].

There are many other underwater robotics applications in the fields of oceanography, biology and wreck exploration. As this thesis has been realized under fundings of the *Délégation Général pour l'Armement*

(DGA), the french military procurement agency, military applications are also heavily considered, for which applications are area inspection and mine sweeping among others. In both the civilian and military fields, it is sometimes necessary to use a group or a swarm of robots that need to cooperate to realize a task. Projects like *SpiceRack* (http://www.cgg.com/default.aspx?cid=5650) realized by *CGG* and *Saudi Aramco* will use a swarm of 3,000 underwater AUVs to realize seismic surveys of the seabed. For this, the AUVs in the swarm need to position themselves in a precise grid pattern, and stay in position despite potentially strong currents. It is important that the robots localize themselves effectively and quantify the error on their position. If the robot is not where "it thinks" it is, the seismic survey could be corrupted, or a robot could collide against another AUV.

In this context, the use of interval methods is relevant as they allow to represent the set of compatible solutions and their uncertainty. Measures from sensors or variables used to describe the state of the robots are often filled with errors. These can be represented is several ways : probabilistic distributions, point clouds, continuous sets, etc. Usually, sensors and actuators are provided with error bounds, which can be represented as intervals. Using these intervals and a set of equations or inequations, interval analysis allows to numerically compute the interval enclosing the solution. Moreover, interval analysis has several advantages. It works for non-linear equations without approximation, and common functions like cos, *exp* or *sqrt* can be used as are. Moreover, the results are guaranteed. No solution can exist outside the bounds of the interval (according to the data and hypothesis considered). If the result of the computation is an empty set, then there is no solution (or there exists a flaw in the conception or realization of the algorithm). Finally, interval methods are a very to solve systems with a much higher number of equations compared to the number of variables, which is particularly useful for a swarm of robots.

1.2 Objectives, hypothesis, constraints and contributions.

- Multiple thesis have been written on solving the localization of one AUV with interval analysis using *SLAM (Simultaneous Localization and Mapping)* with fleeting data [Le Bars, 2011], using set polynomial based solvers and accumulators [Sliwka, 2011], or even using visibility contractors [Guyonneau, 2013]. Yet, none of them considers the problem of localizing several AUVs moving in a swarm. This is therefore the main focus of this thesis.

- We successively consider a group of 2, then 6, then 300, and finally 1,000 AUVs.

- We consider the system on AUVs to be *decentralized* (or *distributed*), meaning each AUV only computes its own localization; in opposition to a *centralized* system where a central unit computes the position of all AUVs.

- We consider that the robots are well spaced, meaning that they might have moved between the time of emission and reception of acoustic waves.

- We consider that the robots have unsynchronized clocks, which is often the case in real applications and complicates the localization problem.

- We consider both *online localization* (the position is computed in real-time as the robot performs its mission) and *offline localization* (the robot performs its whole mission, then the trajectory is reconstructed

after the mission from all the data collected).

- We use an *Inertial Measurement Unit* (or *IMU*, an electronic device that measures and reports on an AUV's velocity, orientation, and gravitational forces, using a combination of accelerometers and gyroscopes, sometimes also magnetometers), to easily obtain the orientation of the robot with good precision. Therefore, this thesis focuses on solving the position of the robots rather than their orientation, considered known.

- In this thesis, we often consider the resolution of differential equations. As a lot of research have already been done on this specific subject [Nedialkov, 2006][Nedialkov et al., 1999], it is not the focus of this thesis. We purposefully use a simple *Euler method* to solve the equations and prove the concept, despite loosing the interval guarantee.

- All developments are done in the *IBEX (Interval Based EXplorer) library* [Chabert and Jaulin, 2009](`http://www.ibex-lib.org/`) for easy reusability and transparency of the results.

1.3 Plan

In this first chapter we introduced the notion of localization and why it is important for autonomous robots, especially underwater.

In the second chapter we present the basic notions of interval arithmetic and a few algorithms to estimate parameters of a given system with observations of this system with bounded errors. Using set inversion via interval analysis, we compute an inner and outer approximation of a static localization problem with outliers.

In the third chapter we extend the contractor algebra to allow for the geometrical transformation of contractors and show that it is possible to build minimal contractors in a very easy way for some constraints with symmetries. As an application, we consider the construction of a contractor associated with the constraint $\theta = atan2\,(x, y)$ and demonstrate its efficiency on another static localization problem.

In the fourth chapter we no longer consider static problems and modelize dynamic systems. We introduce the notion of *tubes*, or *interval of functions* to enclose intervals at different times. Then, an arithmetic is developed around this notion, and a contractor-based approach follows. As a result, a method is proposed to contract tubes that enclose the solution. Several test cases are provided to demonstrate the approach, including the estimation of the position of AUVs.

In the fifth chapter we go further by considering that the clocks of the robots are unsynchronized, making time measurements uncertain. To solve this problem, we consider the cooperative localization problem as an inter-temporal constraints satisfaction problem. We contract the box around the position of the AUVs and their clock using a forward-backward algorithm on each measurement made by the AUVs. A simulator is developed to prove the efficiency of the algorithm and real tests at sea are then made.

Finally, the sixth chapter focuses on studying the scalability of our approach by scaling up our simulation to 1,000 AUVs evolving together in real-time.

The seventh chapter concludes the thesis.

Chapter 2

Interval analysis and its application to robot localization

2.1 Introduction

Interval analysis has become over the past few years a strong alternative to traditional probabilistic approaches [Bonnifait and Garcia, 1996][Lacroix et al., 2002][Thrun et al., 2005] to solve complex non-linear systems of equations, e.g. Simultaneous Localization And Mapping [Di Marco et al., 2001][Jaulin, 2011], [Drocourt et al., 2005], 3D Reconstruction (presented in annex A), path planning [Delanoue et al., 2006], design of robust controllers [Lhommeau et al., 2004], or more generally the characterization of the state evolution of dynamic systems [Aubin and Frankowska., 1990][Abdallah et al., 2008]. The idea is to cast the system as a constraint satisfaction problem by considering intervals enclosing the solution and use contractors (built from the constraints) to successively contract these intervals until a fixed point is reached.

This first chapter introduces the basic notions of interval analysis and demonstrates their application to a simple academic localization problem. The objective is to estimate a vector of parameters from a set of measurements. In other words, we want to find the position of the robots given distance measurements between them. We first consider the problem to be planar and static, meaning that the time is fixed and therefore that the robots are not moving (we will study dynamic systems in the following chapters). The distance measurements are synchronous, meaning that all measurements are obtained at the same sampling time.

Consider the following problem : A robot of unknown Cartesian position measures its distance to other robots, for which their positions are known. This problem is similar to positioning problems involving fixed beacons [Drevelle, 2011]. Consider each robot capable of radio-communication (or acoustic communication underwater) and able to measure distances using time-of-flight technics [Röhrig and Müller, 2009]. The observation model is :

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.1}$$

where (x_i, y_i) are the coordinates of robot i, (x_j, y_j) are the coordinates of robot j and d_{ij} the distance measured between them. Notice that this problem is non-linear, especially when robots are close to each other.

Outliers can appear when the measurement error is too high (e.g. because of interferences) or when the observation model poorly matches the reality. For instance, when robots are indoors, radio waves can reflect on the walls, a phenomenon known as *multi-beam paths*. In some extreme cases, a robot might only receive a reflected wave and therefore measure the reflected distance instead on the true line of sight distance.

Like with beacons, a robot needs to measure the distances to a minimum of three other robots with known positions to localize itself efficiently, a technique known as triangulation. Usually, and especially in swarms, robots can actually collect many more measurements. This redundancy can be used to increase the precision of the localization, and eventually detect and exclude outliers.

This chapter is organized as follows. Section 2 briefly introduces the set-membership approach and section 3 presents the basics of interval analysis. These tools are then used in section 4 to solve the localization problem via set inversion. Section 5 concludes the chapter.

2.2 Set-membership approach

We consider a set-membership approach which is based on bounded errors and consists in computing all the parameters compatible with the observation and their respective error bounds. The bounds on the observation are supposed to be known. Therefore, the problem is not to find the value of the parameters that minimize the error, but to find all values compatible with the observation bounds.

Consider that the error on the distances measured by our robot is bounded. We can write $d_{ij} \in d_{ij}^{measured} + [e_{ij}]$ where d_{ij} is the real distance, $d_{ij}^{measured}$ is the measurement and $[e_{ij}]$ is the interval enclosing the measurement error. The measurements are thus represented by rings centered on each robot. The intersection of the rings encloses the position of the robot making the measurements. Fig. 2.1 demonstrates the principle for one robot measuring its distance to three others.

Notice that the solution set can have any shape. That's why multiple set-membership representations have been developed over the years, each more or less adapted to specific classes of problems. From oldest to newest, we can find interval boxes [Moore, 1966], sub-pavings [Jaulin and Walter, 1993], ellipsoids [Maksarov and Norton, 1996] and polyhedral approximations [Combastel, 2005]. Fig. 2.2 illustrates each representation. However, for non-linear systems, only intervals and sub-pavings have been proven efficient [Jaulin et al., 2001a], which is why this thesis will only consider these.

2.3 Interval analysis

In interval analysis, we consider intervals instead of real numbers. This was first developed to quantify the error on numerical computations [Moore, 1966]. In computers, real numbers are usually represented by floats with limited significant digits. This limitation leads to a small error that can drastically propagate

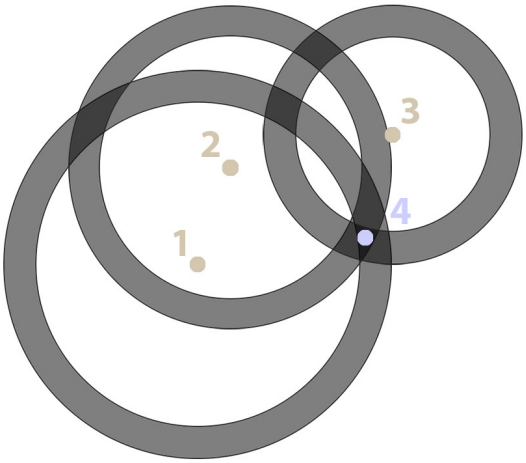

Figure 2.1: The robot 4 (in purple) is localized by measuring distances with bounded errors to three other robots (1, 2 and 3 in brown).

and increase along successive operations. In this thesis, we will use interval analysis to manipulate the uncertainty on parameters. This section presents its basics.

2.3.1 Intervals

An *interval* $[x]$ is defined as the set of numbers x between a lower bound $\underline{x}$ and an upper bound $\bar{x}$.

$$[x] = [\underline{x}, \bar{x}] = \{x \in \mathbb{R}, \underline{x} \leq x \leq \bar{x}\} \tag{2.2}$$

This representation has several advantages: it allows us to represent random variables with imprecise probability density functions, deal with uncertainties in a reliable way and more importantly, it is possible to contract the interval around all feasible values given a set of constraints (i.e. equations or inequalities). For example, the sets $\emptyset = [\infty, -\infty]_{\mathbb{R}}$; $\mathbb{R} = [-\infty, \infty]_{\mathbb{R}}$; $[0, 1]_{\mathbb{R}}$ and $[0, \infty]_{\mathbb{R}}$ are intervals of $\mathbb{R}$, the set $\{2, 3, 4, 5\} = [2, 5]_{\mathbb{N}}$ is an interval of the set of integers $\mathbb{N}$ and the set $\{4, 6, 8, 10\} = [4, 10]_{2\mathbb{N}}$ is an interval of $2\mathbb{N}$. Let us first work with intervals of $\mathbb{R}$.

The *intersection* of two non-empty closed intervals $[x]$ and $[y]$ satisfies:

$$[x] \cap [y] = \begin{cases} [\max\{\underline{x}, \underline{y}\}, \min\{\bar{x}, \bar{y}\}] & \text{if } \max\{\underline{x}, \underline{y}\} \leq \min\{\bar{x}, \bar{y}\} \\ \emptyset & \text{otherwise} \end{cases} \tag{2.3}$$

Example: $[1, 3] \cap [2, 5] = [2, 3]$

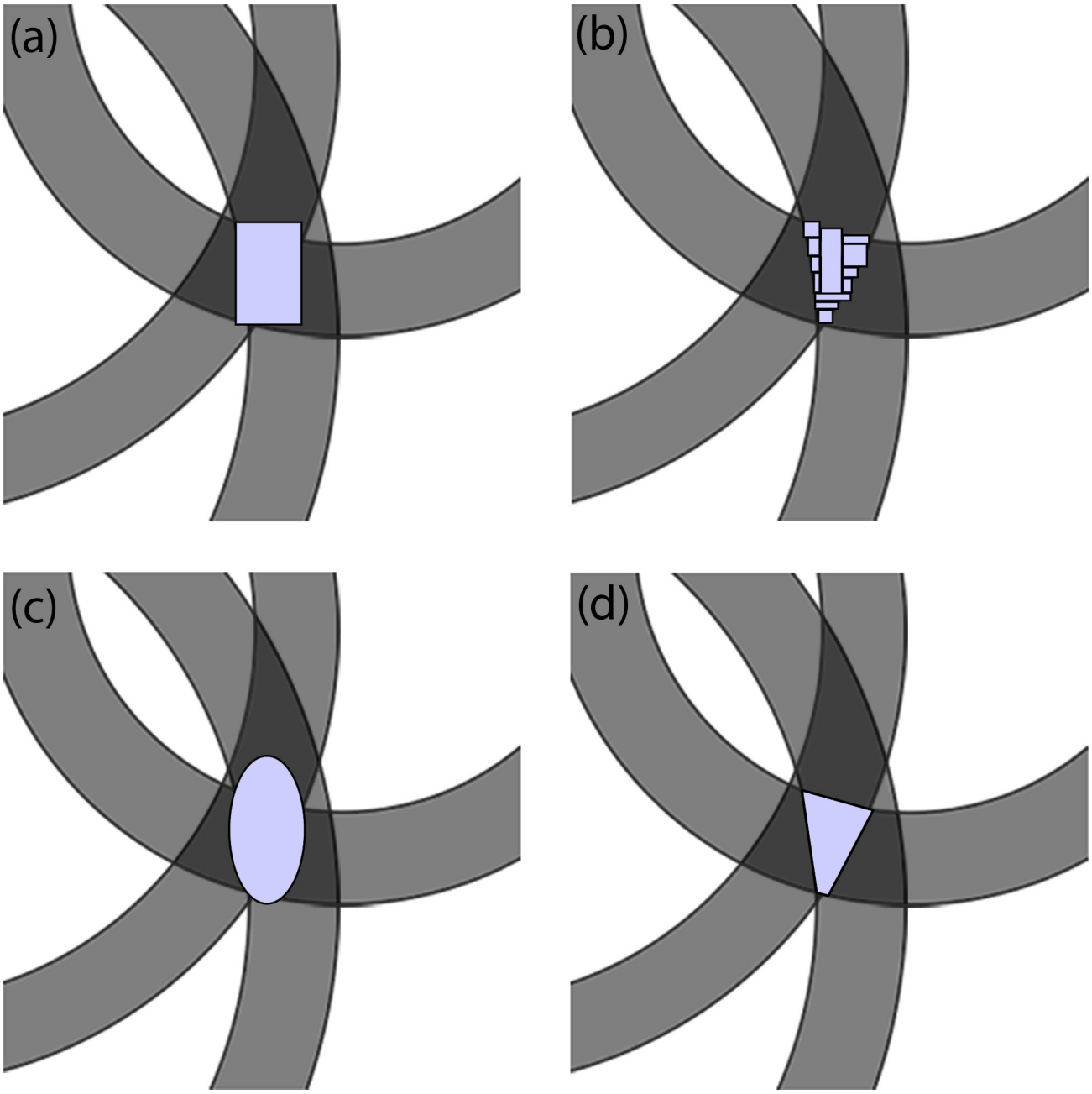

Figure 2.2: Multiple set-membership representations of the solution : (a) Interval box, (b) sub-paving, (c) ellipsoids, (d) polyhedral approximations.

The *interval union* of two non-empty closed intervals $[x]$ and $[y]$ satisfies:

$$[x] \sqcup [y] = [\min\{\underline{x}, \underline{y}\}, \max\{\bar{x}, \bar{y}\}] \tag{2.4}$$

Example: $[1, 3] \sqcup [5, 7] = [1, 7]$

The interval union is not to be mistaken with the simple *union* $\cup$ which might not be an interval. We define the *interval hull* of a subset $\mathbb{X}$ of $\mathbb{R}$ as the smallest interval $[\mathbb{X}]$ that contains it. Therefore:

$$[x] \sqcup [y] = [[x] \cup [y]] \tag{2.5}$$

The *center* of a non-empty closed interval $[x]$ is

$$mid([x]) = \frac{\underline{x}+\bar{x}}{2} \tag{2.6}$$

The *width* of a non-empty interval $[x]$ is

$$w([x]) = \bar{x} - \underline{x} \tag{2.7}$$

A *punctual interval* is denoted by $\{x\} = [x, x]$.

2.3.2 Interval Arithmetic

We can apply arithmetic operators and functions to intervals to obtain all feasible values of the variables. For example, consider a real α and $[x]$ a non-empty interval. Then

$$\alpha[x] = \begin{cases} [\alpha\underline{x}, \alpha\bar{x}] & \text{if } \alpha \geq 0 \\ [\alpha\bar{x}, \alpha\underline{x}] & \text{if } \alpha \leq 0 \end{cases} \tag{2.8}$$

For two intervals $[x]$ and $[y]$ and an operator $\diamond \in \{+, -, *, /\}$, we define $[x] \diamond [y]$ as the smallest interval containing all feasible values for $x \diamond y$ when $x \in [x]$ and $y \in [y]$ or

$$[x] \diamond [y] = [\{x \diamond y \in \mathbb{R} | x \in [x], y \in [y]\}] \tag{2.9}$$

In the case of closed intervals, we have

$$[x] + [y] = [\underline{x} + \underline{y}, \bar{x} - \bar{y}] \tag{2.10}$$

$$[x] - [y] = [\underline{x} - \bar{y}, \bar{x} - \underline{y}] \tag{2.11}$$

$$[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}] \tag{2.12}$$

The inversion is given by

$$1/[y] = \begin{cases} \emptyset & \text{if } [y] = [0,0] \\ [1/\bar{y}, 1/\underline{y}] & \text{if } 0 \notin [y] \\ [1/\bar{y}, \infty[& \text{if } \underline{y} = 0 \text{ and } \bar{y} > 0 \\ ]-\infty, 1/\bar{y}] & \text{if } \underline{y} < 0 \text{ and } \bar{y} = 0 \\ ]-\infty, \infty[& \text{if } \underline{y} < 0 \text{ and } \bar{y} > 0 \end{cases} \tag{2.13}$$

and the division by

$$[x]/[y] = [x] * (1/[y]) \tag{2.14}$$

These rules are simplified for punctual intervals, in which case we use the rules of arithmetic for real numbers. Therefore interval arithmetic can be considered as an extension of the arithmetic for real numbers.

For example:

$$[-1,3] + [2,7] = [1,10] \tag{2.15}$$
$$[-1,3] - [2,7] = [-8,1] \tag{2.16}$$
$$[-1,3].[2,7] = [-7,21] \tag{2.17}$$
$$[-1,3]/[2,7] = [-1/2, 3/2] \tag{2.18}$$

2.3.3 Boxes

A *box* $[\mathbf{x}]$ of $\mathbb{R}^n$ is the Cartesian product of n intervals. The set of all boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$.

$$[\mathbf{x}] = [x_1] \times ... \times [x_n] = [\underline{x}_1, \bar{x}_1] \times ... \times [\underline{x}_n, \bar{x}_n] \tag{2.19}$$

Fig. 2.3 represents an axis-aligned box of $\mathbb{R}^n$.

Most rules defined for intervals are compatible with boxes. Thus, upper and lower bounds for a box are

$$\underline{x} = (\underline{x}_1, ..., \underline{x}_n)^T \tag{2.20}$$
$$\bar{\mathbf{x}} = (\bar{x}_1, ..., \bar{x}_n)^T \tag{2.21}$$

The *width of a box* is defined by

$$w([\mathbf{x}]) = \max_{1 \leq i \leq n} w([x_i]) \tag{2.22}$$

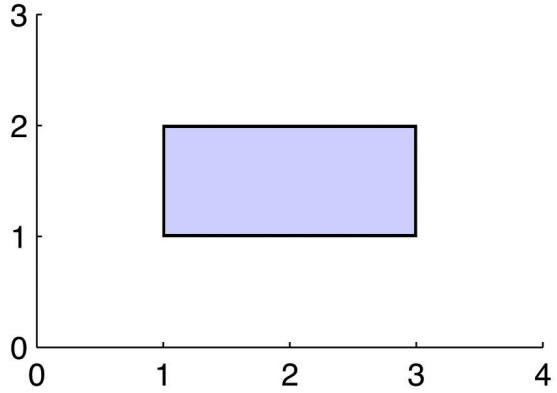

Figure 2.3: A box $[1,3] \times [1,2]$

Finally, operations on intervals can be extended to operations on boxes by considering interval computations on each component of the box.

2.3.4 Inclusion function

The image of $f([x])$ of an interval by a function f is

$$f([x]) = \{f(x) | x \in [x]\} \tag{2.23}$$

This image might not be an interval. Indeed, if f is not continuous $f([x])$ is a union of intervals. The interval extension is defined as the function returning the following interval hull:

$$[f]([x]) = [\{f(x) | x \in [x]\}] \tag{2.24}$$

The *interval extension* of elementary functions can be directly written through its bounds. E.g., for a non-empty interval $[x]$, the interval extension of the exponential function is

$$[\exp]([x]) = [\exp \underline{x}, \exp \bar{x}] \tag{2.25}$$

In case of non-monotonic functions, the situation is more complex. Indeed $[\cos]([-\pi, \pi]) = [-1, 1]$ differs from $[\cos(-\pi), \cos(\pi)] = [-1, -1]$. Specific algorithms can be constructed for the interval evaluation of such functions [Bouron, 2002].

In the same way, the image of a box $[\mathbf{x}]$ by a function $\mathbf{f}$ is usually not a box, and there is no both exact and easy representation of the image set. We will therefore use *inclusion functions*, which enclose the image by a box. The interval function $[\mathbf{f}] : \mathbb{IR}^n \to \mathbb{IR}^m$ is an inclusion function for $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ if and only if

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathbf{f}([\mathbf{x}]) \subset [\mathbf{f}]([\mathbf{x}]) \tag{2.26}$$

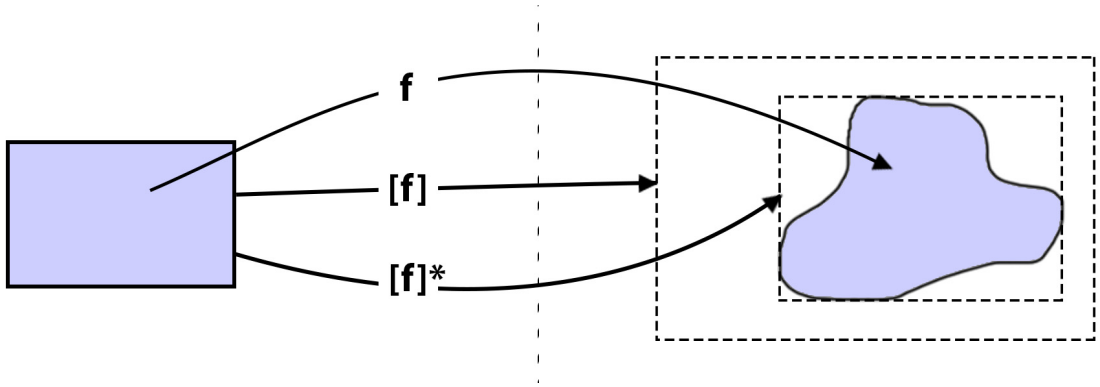

Figure 2.4: Image of a box by the function $\mathbf{f}$, the inclusion function $[\mathbf{f}]$ and the minimal inclusion function $[\mathbf{f}]^*$.

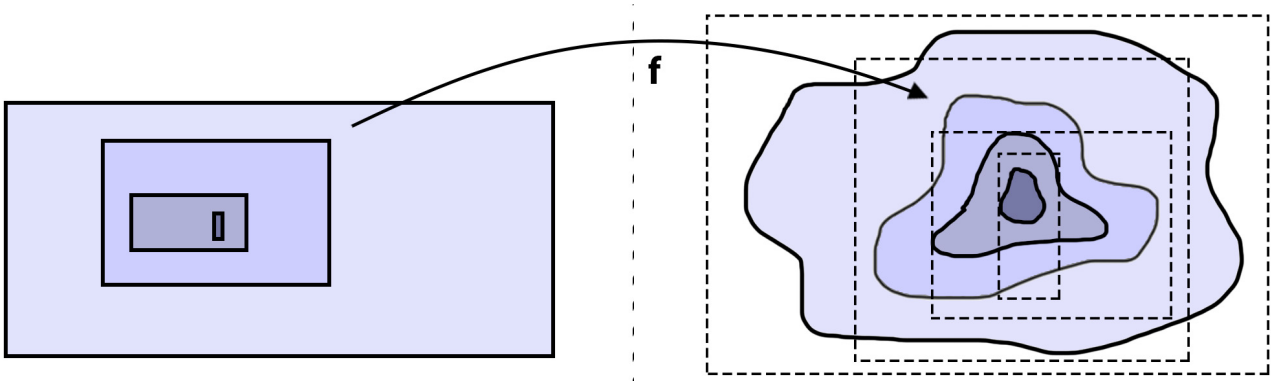

Figure 2.5: Example of both convergent and monotonic inclusion function.

In other words, for all shapes of $\mathbf{f}([\mathbf{x}])$, $[\mathbf{f}]([\mathbf{x}])$ contains the image of $[\mathbf{x}]$ by $\mathbf{f}$. Fig. 2.4 illustrates this notion.

An inclusion function is *thin* if the image of a punctual interval $\{\mathbf{x}\}$ is a punctual interval, i.e. if $[\mathbf{f}](\{\mathbf{x}\}) = \mathbf{f}(\{\mathbf{x}\})$.

An inclusion function is *convergent* if for a series of boxes $[\mathbf{x}](k)$, we have

$$\lim_{k \longrightarrow \infty} w([\mathbf{x}](k)) = 0 \Longrightarrow \lim_{k \longrightarrow \infty} w([\mathbf{f}]([\mathbf{x}](k))) = 0 \tag{2.27}$$

Convergence is usually a requirement for algorithms using interval analysis.

An inclusion function is *monotonic* in respect with the inclusion if $[\mathbf{x}] \subset [\mathbf{y}] \Longrightarrow [\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{f}]([\mathbf{y}])$.

An inclusion function is *minimal* if, for all $[\mathbf{x}]$, $[\mathbf{f}]([\mathbf{x}])$ is the smallest box enclosing $\mathbf{f}([\mathbf{x}])$. This inclusion function is unique and is noted $[\mathbf{f}]^*$. All inclusion functions that are not the minimal inclusion functions are called *pessimistic* because of the poor enclosure of the image.

Consider a function f built as a finite number of compositions of elementary functions (e.g. *sin*, *cos*, *sqr*, *max*, etc.) and operators $(+, -, /, *)$. A simple method to build an inclusion function for f is by replacing the scalar variables by their interval counterpart and replacing the elementary functions by their interval

extension. The function obtained is called the *natural inclusion function* and is denoted $[f]_n$. It is finite and monotonic in respect to the inclusion. Moreover, if f is solely made of continuous elementary functions, then $[f]_n$ is convergent.

Example : Consider $f(x,y) = x*y+\cos(x+y)-\sqrt{x+2}$. The natural inclusion function of f is $[f]_n([x],[y]) = [x]*[y]+\cos([x]+[y])-\sqrt{[x]+2}$.

Natural inclusion functions are usually not minimal because of dependencies between variables and the *wrapping effect*. However, the natural inclusion function will be minimal if each variable appears only once in the definition of f and if all operators and elementary functions used are continuous.

In our localization example with four robots, the observation function is :

$$\begin{array}{ccc} \mathbf{g} : \mathbb{R}^2 & \rightarrow & \mathbb{R}^3 \\ \begin{pmatrix} x_4 \\ y_4 \end{pmatrix} & \longmapsto & \begin{pmatrix} \sqrt{(x_4-x_1)^2+(y_4-y_1)^2} \\ \sqrt{(x_4-x_2)^2+(y_4-y_2)^2} \\ \sqrt{(x_4-x_3)^2+(y_4-y_3)^2} \end{pmatrix} \end{array} \tag{2.28}$$

The natural inclusion function for $\mathbf{g}$ is :

$$\begin{array}{ccc} [\mathbf{g}] : \mathbb{IR}^2 & \rightarrow & \mathbb{IR}^3 \\ \begin{pmatrix} [x_4] \\ [y_4] \end{pmatrix} & \longmapsto & \begin{pmatrix} \sqrt{([x_4]-[x_1])^2+([y_4]-[y_1])^2} \\ \sqrt{([x_4]-[x_2])^2+([y_4]-[y_2])^2} \\ \sqrt{([x_4]-[x_3])^2+([y_4]-[y_3])^2} \end{pmatrix} \end{array} \tag{2.29}$$

Notice that for now, we consider the position of robots 1,2 and 3 as punctual intervals (i.e. $[x_1] = \{x_1\}$). Moreover, as each variable appears only once and as we only use continuous elementary functions and operators, $[\mathbf{g}]$ is the minimal inclusion function for $\mathbf{g}$.

2.3.5 Contractors

Consider n_x real variables $x_i \in \mathbb{R}, i \in \{1,...,n_x\}$ linked by n_f relations (or *constraints*) of the form :

$$f_j(x_1,x_2,...,x_{n_x}) = 0, j \in \{1,...,n_f\} \tag{2.30}$$

whereby f_j denotes the function for each coordinate j. We know that each variable x_i belong to a domain $\mathbb{X}_i$. To simplify, we consider the domains as intervals noted $[x_i]$. We define $\mathbf{x} = (x_1,x_2,...,x_{n_x})^T$ and the domain for $\mathbf{x}$ as $[\mathbf{x}] = [x_1] \times [x_2] \times ... \times [x_{n_x}]$. We also note that $\mathbf{f}$ is a function with coordinate functions

f_j. We can therefore re-write (2.30) in a vector form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ and this represents a *constraint satisfaction problem* (*CSP*) that we can call $\Im$ and write

$$\Im : (\mathbf{f}(\mathbf{x}) = 0, x \in [x]) \tag{2.31}$$

Therefore a CSP is composed of a set of variables, domains containing these variables, and constraints. The solution $\mathbb{S}$ of $\Im$ is defined as

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}] | \mathbf{f}(\mathbf{x}) = \mathbf{0}\} \tag{2.32}$$

Contracting a CSP $\Im$ consists in replacing the domain $[\mathbf{x}]$ with a smaller domain $[\mathbf{x}']$ without changing the solution set. We have $\mathbb{S} \subset [\mathbf{x}'] \subset [\mathbf{x}]$. An operator that allows the contraction of $\Im$ is called a *contractor*. We define the *minimal contractor* as the contractor replacing $[\mathbf{x}]$ by the smallest box containing $\mathbb{S}$.

Many problems of estimation, control, robotics, etc. can be represented as constraint satisfaction problems [Araya et al., 2008][Ceberio and Granvilliers, 2001] and many contractors can be designed to contract the domains more of less well depending on the class of the problem [Chabert and Jaulin, 2009] : Gauss elimination, *Gauss-Seidel* algorithm, *Krawczyk* method, *Newton* algorithm, etc. [Jaulin et al., 2001a]. The one we are going to use in our localization problem is the *forward-backward contractor* (also known as *HC4-Revise*) [Benhamou et al., 1999] which contracts the domains of the CSP $\Im : (\mathbf{f}(\mathbf{x}) = 0, x \in [x])$ by isolating each constraint separately. We suppose that each constraint has the form $f_j(x_1, x_2, ..., x_{n_x}) = 0$, and that the function f_j can be broken down into a series of operations involving operators and elementary functions such as $+, -, *, /, sin, cos, exp, etc.$ called *primary constraints*.

In our localization problem, the associated constraint with the distance measurement is written:

$$d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.33}$$

It can therefore be broken down into primary constraints by introducing intermediate variables:

$$\begin{aligned} i_1 &= -x_j \\ i_2 &= x_i + i_1 \\ i_3 &= i_2^2 \\ i_4 &= -y_j \\ i_5 &= y_i + i_4 \\ i_6 &= i_5^2 \\ i_7 &= i_3 + i_6 \\ d &= \sqrt{i_7} \end{aligned} \tag{2.34}$$

The initial domains associated with the intermediate variables i_k are $]-\infty; \infty[$. A method to contract $\Im$ with the constraint is to contract each primitive constraint until the contractor reaches a fixed point.

This is the principle of constraint propagation introduced by Waltz [Waltz, 1975]. For constraints involving two variables and a function, such as the square root, two steps of contraction are made by rewriting the constraint: one from the direct image of the function and one from the inverse. E.g. the constraint $d = \sqrt{i_7}$ can be re-written in two forms:

$$d = \sqrt{i_7} \tag{2.35}$$
$$i_7 = d^2 \tag{2.36}$$

and the contraction steps are :

$$[d] = [d] \cap \sqrt{[i_7]} \tag{2.37}$$
$$[i_7] = [i_7] \cap [d^2] \tag{2.38}$$

For constraints linking three variables with a binary operation such as an addition, there are three ways to rewrite the constraint. Let's consider the constraint $i_7 = i_3 + i_6$ with for example the initial intervals $[i_3] = [-\infty, 2], [i_6] = [-\infty, 3]$ and $[i_7] = [4, \infty]$. We can easily contract these intervals without removing any feasible value:

$$\begin{aligned} i_7 = i_3 + i_6 \rightarrow i_7 &\in [4, \infty] \cap ([-\infty, 2] + [-\infty, 3]) \\ &= [4, \infty] \cap [-\infty, 5] = [4, 5] \end{aligned} \tag{2.39}$$
$$\begin{aligned} i_3 = i_7 - i_6 \rightarrow i_3 &\in [-\infty, 2] \cap ([4, \infty] - [-\infty, 3]) \\ &= [-\infty, 2] \cap [1, \infty] = [1, 2] \end{aligned} \tag{2.40}$$
$$\begin{aligned} i_6 = i_7 - i_3 \rightarrow z &\in [-\infty, 3] \cap ([4, \infty] + [-\infty, 2]) \\ &= [-\infty, 3] \cap [2, \infty] = [2, 3] \end{aligned} \tag{2.41}$$

We obtain smaller intervals: $[i_3] = [1, 2], [i_6] = [2, 3]$ and $[i_7] = [4, 5]$.

The same principle is applied for all primary constraints in (2.34) in order to contract the intervals around the feasible values of (2.33). The sequence of contractions made by the forward-backward algorithm is optimal to maximize the contraction. The forward-backward algorithm is presented in table 2.1 and runs successively on each constraint, i.e. each distance measurement. The contractor is minimal for each constraint but dependencies between constraints might make it so that the contractions are not minimal for the whole system. To contract the CSP further, we can run the contractor again and again until a fixed point is reached.

A similar approach have been used in [Gning and Bonnifait, 2006] to localize a real car.

Fig. 2.6 presents the successive contractions that occur when calling the forward-backward contractor associated with each distance measurement until a fixed point is reached.

	Algorithm C_{FB} (in : box, inout : $[x_i], [x_j], [y_i], [y_j], [d]$)
	// Forward steps
1	$[i_1] := [i_1] \cap -[x_j]$
2	$[i_2] := [i_2] \cap ([x_i] + [i_1])$
3	$[i_3] := [i_3] \cap [i_2]^2$
4	$[i_4] := [i_4] \cap [-y_j]$
5	$[i_5] := [i_5] \cap ([y_i] + [i_4])$
6	$[i_6] := [i_6] \cap [i_5]^2$
7	$[i_7] := [i_7] \cap ([i_3] + [i_6])$
8	$[d] := [d] \cap \sqrt{[i_7]}$
	// Backward steps
9	$[i_7] := [i_7] \cap [d]^2$
10	$[i_3] := [i_3] \cap ([i_7] - [i_6])$
11	$[i_6] := [i_6] \cap ([i_7] - [i_3])$
12	$[i_5] := [i_5] \cap \sqrt{[i_6]}$
13	$[y_i] := [y_i] \cap ([i_5] - [i_4])$
14	$[i_4] := [i_4] \cap ([i_5] - [y])$
15	$[y_j] := [y_j] \cap -[i_4]$
16	$[i_2] := [i_2] \cap \sqrt{[i_3]}$
17	$[x_i] := [x_i] \cap ([i_2] - [i_1])$
18	$[i_4] := [i_4] \cap ([i_2] - [x_i])$
19	$[x_j] := [x_j] \cap -[i_1]$

Table 2.1: Forward-backward algorithm applied to distance measurement (2.33).

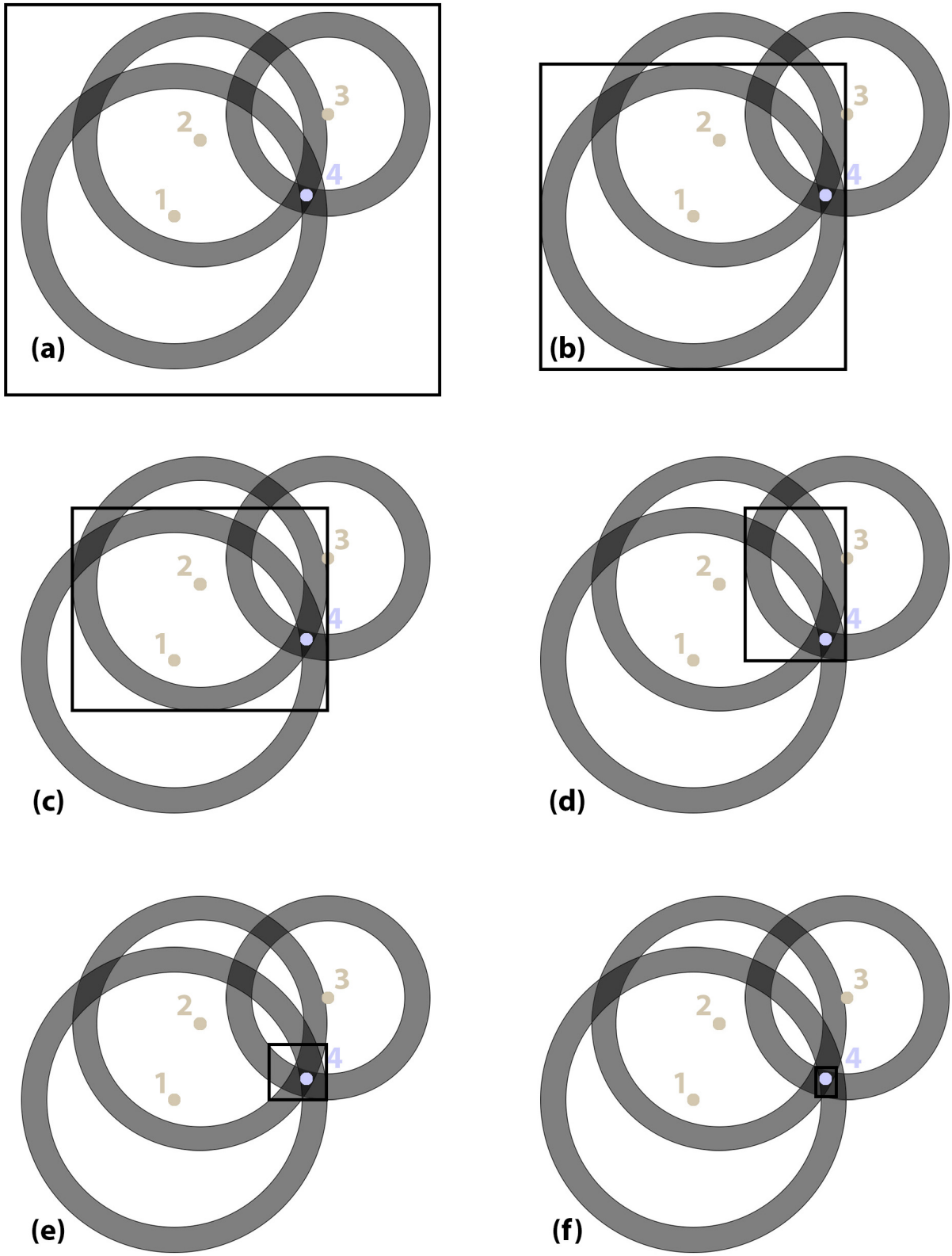

Figure 2.6: Successive contractions of the box enclosing the position of robot 4, (a) at start, (b) after applying the forward-backward contractor with robot 1, (c) with robot 2, (d) with robot 3, (e) with robot 1, 2 and 3 again and (f) until a fixed point is reached.

Let us consider another example involving three constraints:

$$(C_1) \quad : \quad y = x^2 \tag{2.42}$$
$$(C_2) \quad : \quad xy = 2 \tag{2.43}$$
$$(C_3) \quad : \quad y = -x + 1 \tag{2.44}$$

If we have no information about x and y, we assign them the domain $[-\infty, \infty]$. Then, we contracts these domains by applying the contractors until the contractions a not significant anymore. Notice that the resulting domains are not dependent on the order in which we apply the contractors. However, computation time might be. Let's apply them in the following order:

$$(C_1) \rightarrow y \quad \in \quad [-\infty, \infty]^2 = [0, \infty] \tag{2.45}$$
$$(C_2) \rightarrow x \quad \in \quad 2/[0, \infty] = [0, \infty] \tag{2.46}$$
$$(C_3) \rightarrow y \quad \in \quad [0, \infty] \cap ((-3).[0, \infty] + 1) = [0, 1] \tag{2.47}$$
$$x \in [0, \infty] \cap (-[0, 1]/3 + 1/3) = [0, 1/3]$$
$$(C_1) \rightarrow y \quad \in \quad [0, 1] \cap [0, 1/3]^2 = [0, 1/9] \tag{2.48}$$
$$(C_2) \rightarrow x \quad \in \quad [0, 1/3] \cap 1/[0, 1/9] = \emptyset \tag{2.49}$$
$$y \in [0, 1/9] \cap 1/\emptyset = \emptyset$$

We obtain empty intervals which means that there is no feasible values for x and y that satisfy the system.

2.4 Set Inversion Via Interval Analysis (SIVIA)

Intervals and boxes are easy to manipulate thanks to interval arithmetic and inclusion functions. Contractors, as defined in the previous section, contract boxes according to given constraints. However, the form of the solution set might be complex and its enclosure by a simple box might not be satisfactory, especially if the solution set is dissociated.

2.4.1 Sub-paving

In order to represent a complex set $\mathbb{X}$ accurately while taking advantage of interval methods, we use *sub-pavings* of $\mathbb{R}^n$. A sub-paving of a box $[\mathbf{x}] \subset \mathbb{R}^n$ is the union of non-empty non overlapping boxes. Two boxes of the same sub-paving can have a non-empty intersection if they have a common border, but their inside have to be disjoint.

As presented in Fig. 2.7, we can enclose the set $\mathbb{X}$ in-between two sub-pavings : an inner approximation $\underline{\mathbb{X}}$ and an outer approximation $\bar{\mathbb{X}} = \underline{\mathbb{X}} \cup \Delta\mathbb{X}$, where $\Delta\mathbb{X}$ is the sub-paving of the border, such that:

$$\underline{\mathbb{X}} \subset \mathbb{X} \subset \bar{\mathbb{X}} \tag{2.50}$$

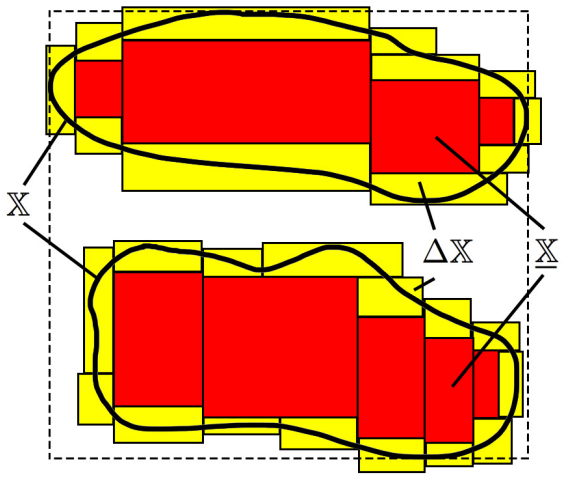

Figure 2.7: Inner approximation $\underline{\mathbb{X}}$ and outter approximation $\bar{\mathbb{X}} = \underline{\mathbb{X}} \cup \Delta\ \mathbb{X}$ of the set $\mathbb{X}$. The dashed square represents the simple minimal box enclosing the set. Sub-pavings show to be more powerful.

Knowing $\underline{\mathbb{X}}$ and $\bar{\mathbb{X}}$ gives us powerful information about $\mathbb{X}$, e.g. if $\underline{\mathbb{X}}$ is non-empty then $\mathbb{X}$ is non-empty as well, and if $\bar{\mathbb{X}}$ is empty then $\mathbb{X}$ is empty as well. The volume of the sub-paving enclosing $\Delta\mathbb{X}$ characterize the precision of the approximation.

When each box of a sub-paving can be obtained with successive bisections, the sub-paving is called *regular*. Regular sub-pavings have multiple advantages including their representation as a binary tree in machines, which limits their storage in memory and make it easy to manipulate them for operations like the union, the intersection or the inclusion [Jaulin et al., 2001b].

2.4.2 SIVIA

We want to characterize the set $\mathbb{X}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ where $\mathbf{f} : \mathbb{X} \longrightarrow \mathbb{Y}$, i.e. we want to characterize the inverse image $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$ by considering $\mathbb{Y}$ as a sub-paving. The *SIVIA* (*Set Inversion Via Interval Analysis*) algorithm presented in table 2.2 computes an inner and outer approximation $\underline{\mathbb{X}}$ and $\bar{\mathbb{X}}$ for $\mathbb{X}$ [Jaulin and Walter, 1993]. Notice that as the algorithm works with boxes, a convergent inclusion function $[\mathbf{f}]$ for $\mathbf{f}$ is needed. The algorithm works as follows :

- A starting box $[\mathbf{x}]$ is provided, and is guaranteed to enclose the solution. As we may have no specific knowledge about the solution, the box provided can be very large.

- An inclusion test is performed on $[\mathbf{x}]$ using $[\mathbf{f}]$.

· If $[\mathbf{f}]([\mathbf{x}]) \cap [\mathbf{y}] = \emptyset$ then $\mathbf{f}([\mathbf{x}]) \cap [\mathbf{y}] = \emptyset$. In other words, if the image $[\mathbf{x}]$ by $[\mathbf{f}]$ is disjoint from $\mathbb{Y}$, then $[\mathbf{x}]$ is guaranteed not to belong to the solution set. The box is discarded. This case is presented in fig.

	Algorithm $SIVIA$ (in : $[\mathbf{x}], \mathbf{f}, [\mathbf{y}], \varepsilon$; inout: $\underline{\mathbb{X}}, \bar{\mathbb{X}}$)
1	if $[\mathbf{f}]([\mathbf{x}]) \cap [\mathbf{y}] = \emptyset$ then
2	draw($[\mathbf{x}]$, 'blue'); return
3	end if
4	if $[\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{y}]$ then
5	$\underline{\mathbb{X}} = \underline{\mathbb{X}} \cup [\mathbf{x}]$; $\bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw($[\mathbf{x}]$, 'red'); return
6	end if
7	if $w([\mathbf{x}]) < \varepsilon$ then
8	$\bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw ($[\mathbf{x}]$, 'yellow'); return
9	end if
10	bisect $[\mathbf{x}]$ into $[\mathbf{x}_1]$ and $[\mathbf{x}_2]$
11	$SIVIA([\mathbf{x}_1], \mathbf{f}, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$
12	$SIVIA([\mathbf{x}_2], \mathbf{f}, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$

Table 2.2: SIVIA

2.8(a).

· If $[\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{y}]$ then $\mathbf{f}([\mathbf{x}]) \subset [\mathbf{y}]$, i.e. if the image $[\mathbf{x}]$ by $[\mathbf{f}]$ is included in $\mathbb{Y}$, then $[\mathbf{x}]$ is guaranteed to belong to the solution set. The box is added to the sub-paving of $\underline{\mathbb{X}}$ and $\bar{\mathbb{X}}$. This case is presented in fig. 2.8(b).

· For all other cases, $[\mathbf{x}]$ is considered undetermined and bisected into two sub-boxes on which SIVIA is run again. This case is presented in fig. 2.8(c).

- The algorithm is run recursively until the undetermined boxes reach a given width ε.

Fig. 2.9(a) presents the result of the SIVIA algorithm applied to a distance measurement $d_{43} = [3, 4]$ to robot 3 at position $(3, 3)$. The red sub-paving represents the inner approximation $\underline{\mathbb{X}}$ for the solution and the union of the red and yellow sub-pavings represent the outer approximation $\bar{\mathbb{X}}$ for the solution. The blue sub-paving is guaranteed not to contain the solution.

2.4.3 SIVIA with a contractor

bisections are costly in terms of computation time. In order to improve it, we can use a version of SIVIA using a contractor C that contracts the box $[\mathbf{x}]$ around the solution before applying the bisection. For a given number of bisections, the sub-paving obtained is more precise than with a simple SIVIA algorithm. Multiple contractors can be used for C as described in the previous section.

The table 2.3 presents the algorithm and fig. 2.9(b) shows the new sub-paving. Notice that the sub-paving obtained is no more regular, but that every box is now touching the border $\Delta\mathbb{X}$, which shows that the contractor we use is minimal (also called optimal).

Let us now apply the algorithm to all distance measurements in our localization problem. As a reminder, we are looking to enclose the position of robot 4 in a sub-paving knowing the distance to robot 1, 2 and 3.

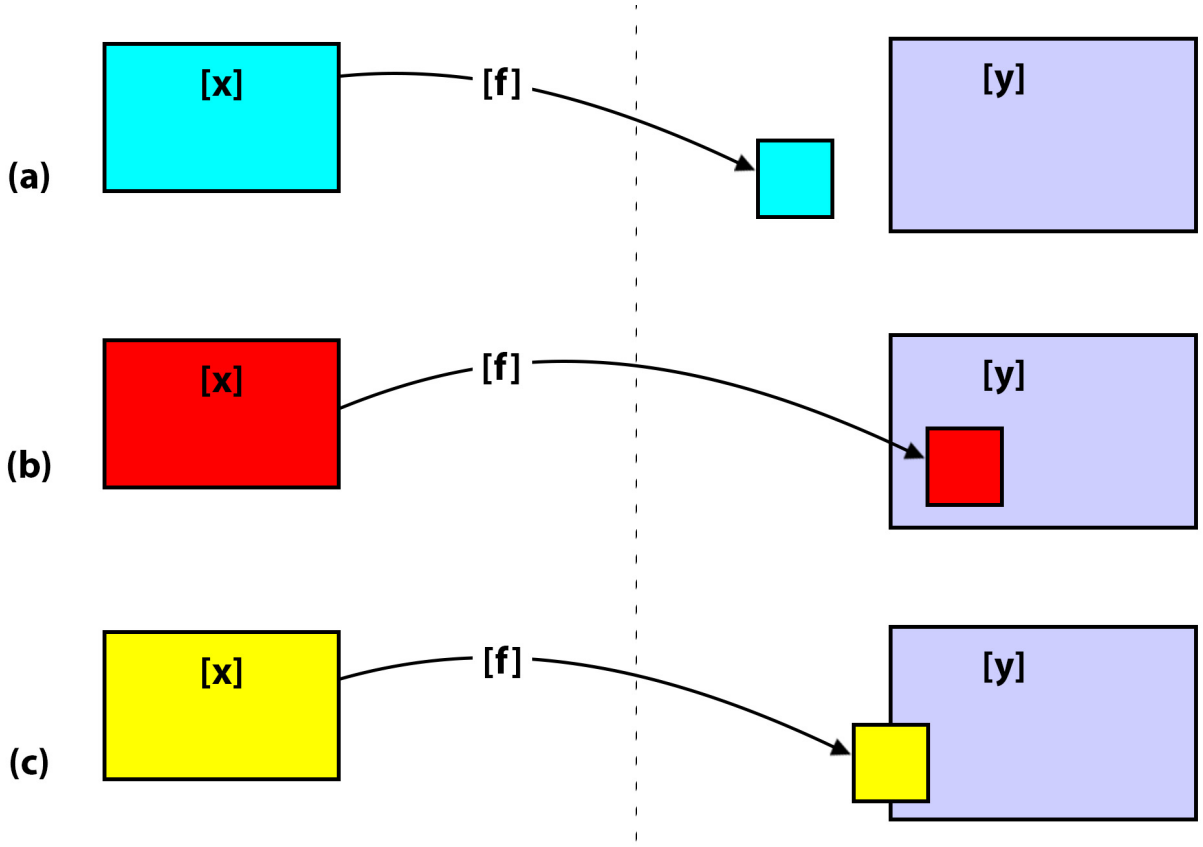

Figure 2.8: Inclusion test for set inversion.

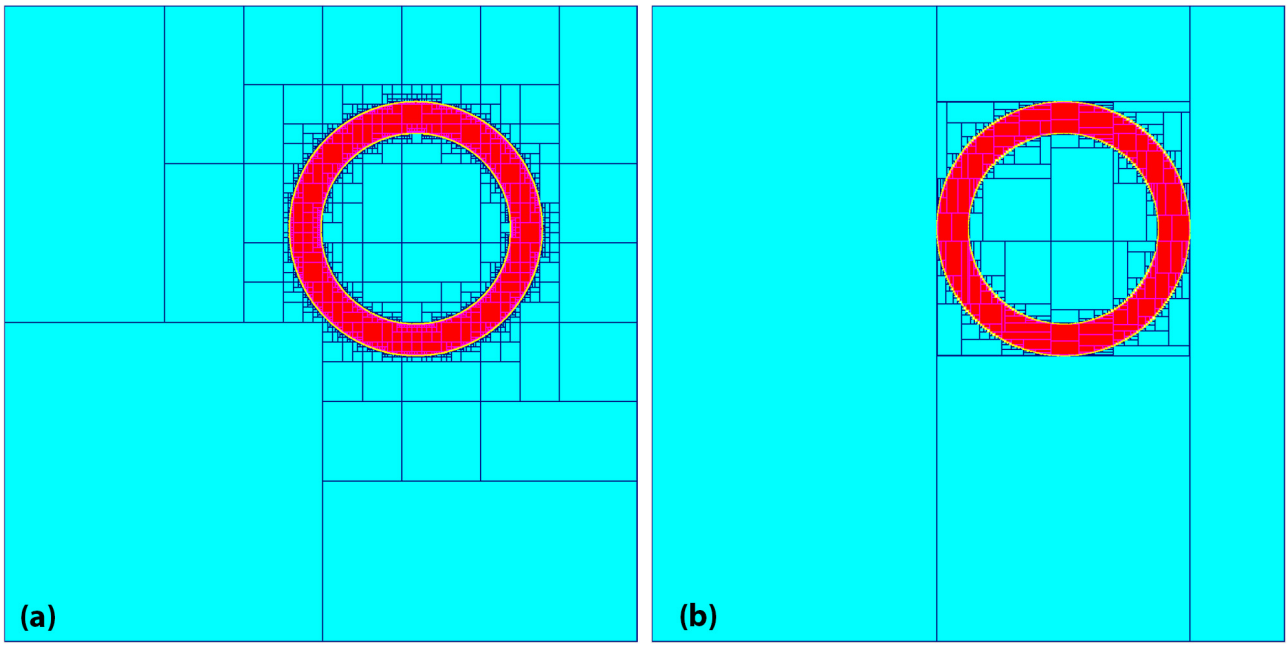

Figure 2.9: (a) Simple SIVIA algorithm applied to a distance measurement $d_{34} = [3, 4]$ to robot 3 at position $(3, 3)$. (b) The SIVIA algorithm with a contractor needs less bissections.

	Algorithm $CSIVIA$ (in : $[\mathbf{x}], C, \mathbf{f}, [\mathbf{y}], \varepsilon$; inout: $\underline{\mathbb{X}}, \bar{\mathbb{X}}$)
1	$[\mathbf{x}] := C([\mathbf{x}])$
2	if $[\mathbf{x}] = \emptyset$ then
3	return
4	end if
5	if $[\mathbf{f}]([\mathbf{x}]) \cap [\mathbf{y}] = \emptyset$ then
6	draw($[\mathbf{x}]$, 'blue'); return
7	end if
8	if $[\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{y}]$ then
9	$\underline{\mathbb{X}} = \underline{\mathbb{X}} \cup [\mathbf{x}]$; $\bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw($[\mathbf{x}]$, 'red'); return
10	end if
11	if $w([\mathbf{x}]) < \varepsilon$ then
12	$\bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw ($[\mathbf{x}]$, 'yellow'); return
13	end if
14	bisect $[\mathbf{x}]$ into $[\mathbf{x}_1]$ and $[\mathbf{x}_2]$
15	$CSIVIA([\mathbf{x}_1], C, \mathbf{f}, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$
16	$CSIVIA([\mathbf{x}_2], C, \mathbf{f}, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$

Table 2.3: SIVIA

Fig. 2.10(a) is the result of $CSIVIA$ to a distance measurement $d_{43} = [3, 4]$ to robot 3 at position $(3, 3)$. Fig. 2.10(b) adds the measurement $d_{41} = [5, 6]$ to robot 1 at position $(-3, -1)$ and fig. 2.10(c) adds the measurement $d_{42} = [4, 5]$ to robot 1 at position $(-2, 2)$. The resulting sub-paving is small enough to localize robot 4 with a good precision.

2.4.4 Robust SIVIA

Notice that the more measurements we add, the better the localization. In general, interval methods are very powerful when the number of constraints is far superior to the number of variables. Let us add a fourth measurement $d_{45} = [3, 4]$ to a robot 5 at position $(4, -4)$. The resulting sub-paving is presented in fig. 2.11(a) and is smaller than in fig. 2.10(c). However, if the measurement is false (e.g. faulty sensor, error bounds too tight, etc.), the resulting sub-paving is empty, meaning that the constraints are incompatible with the system and that no solution exists. If we measure $d_{45} = [1, 2]$, the fig. 2.11(b) shows that there is no solution. This usually happens when using cheap sensors with no redundancy, or when the observation model does not match the real system.

A solution for solving this class of problems is to use the *q-intersection* [Sliwka, 2011]. The principle is to consider that we have q outliers among our m observations (here distance measurements). We then realize a *q-relaxed set inversion* by considering the solutions that only satisfy $m - q$ observations.

Consider m constraints i associated with m solution sets $\mathbb{X}_1, ..., \mathbb{X}_m$ of $\mathbb{R}^n$. The q-intersection is noted $\bigcap_{i \in \{1,...,m\}}^{\{q\}} \mathbb{X}_i$ and is the set of all $x \in \mathbb{R}^n$ that belongs to at least $m - q$ sets $\mathbb{X}_i$. Notice that the 0-relaxed intersection is the classical intersection between sets, and that the $(m-1)$-relaxed intersection is the union

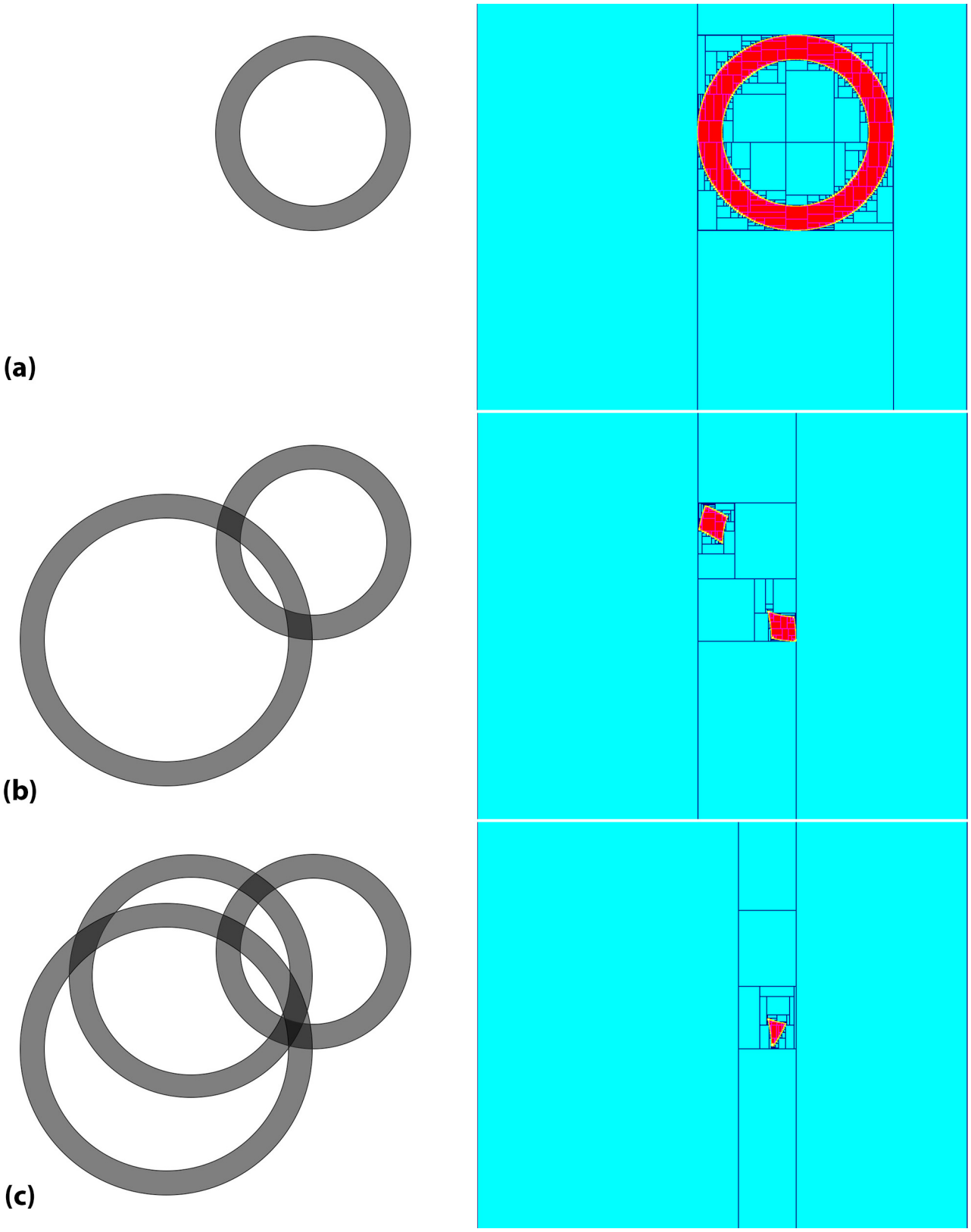

Figure 2.10: SIVIA with a contractor applied to the distance measurement, (a) with robot 3 only, (b) with robot 1 and 3, (c) with robot 1, 2 and 3.

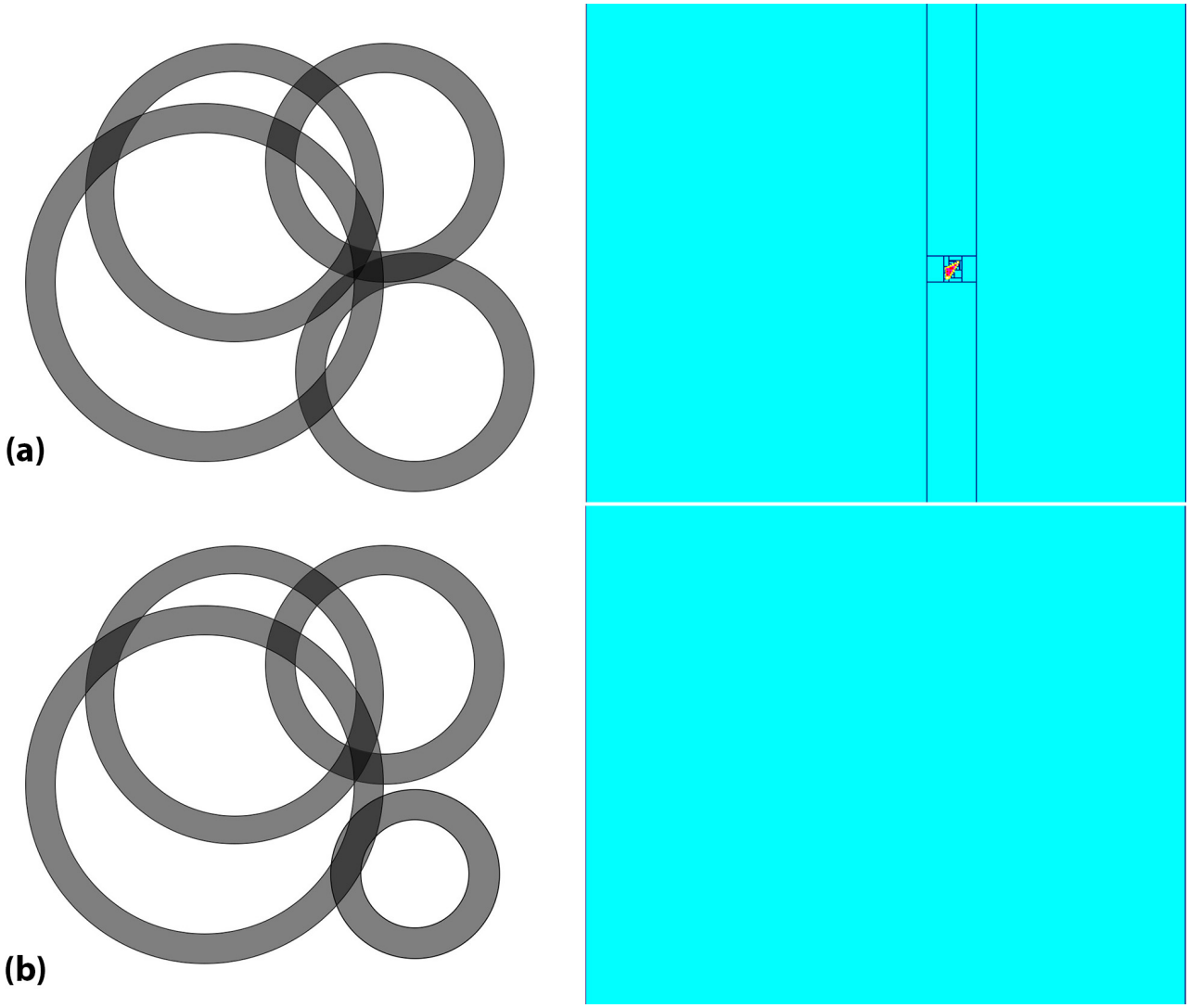

Figure 2.11: (a) A fourth measurement can improve the quality of the localization... (b) unless the measurement is faulty. The subpavings of the inner and outter approximations are empty.

of the m sets. The *RSIVIA* (*Robust Set Inversion Via Interval Analysis*) algorithm [Jaulin et al., 2001b] is presented in table 2.4 and computes the q-relaxed intersection of the solution sets $\mathbb{X}_i$ such that $[\mathbf{x}_i] = f^{-1}([\mathbf{y}_i])$. The inclusion test of *RSIVIA* is slightly different from *SIVIA* to determine the membership of a box to the q-released solution set. We define the *degree of inclusion incl* of a box in another and the *degree of separation sep* between two boxes [Jaulin et al., 2002] such that:

$$incl([\mathbf{x}],[\mathbf{y}]) = \sum_{i=1}^{m} incl([x_i],[y_i]) \tag{2.51}$$

$$sep([\mathbf{x}],[\mathbf{y}]) = \sum_{i=1}^{m} sep([x_i],[y_i]) \tag{2.52}$$

where:

$$\begin{aligned} incl([x_i],[y_i]) &= \begin{cases} 1 & \text{if } [x_i] \subset [y_i] \\ 0 & \text{else} \end{cases} \\ sep([x_i],[y_i]) &= \begin{cases} 1 & \text{if } [x_i] \cap [y_i] = \emptyset \\ 0 & \text{else} \end{cases} \end{aligned} \tag{2.53}$$

We can then define an inclusion test of a box $[\mathbf{x}]$ to the q-relaxed solution set as:

$$\begin{cases} sep([\mathbf{f}]([\mathbf{x}]),[\mathbf{y}]) > q & [\mathbf{x}] \text{ does not contain the solution} \\ incl([\mathbf{f}]([\mathbf{x}]),[\mathbf{y}]) \geq m-q & [\mathbf{x}] \text{ is included in the solution set} \\ else & [\mathbf{x}] \text{ is undetermined} \end{cases} \tag{2.54}$$

Obviously, *RSIVIA* can be combined with *CSIVIA* for a robust set inversion with contractors. Fig. 2.12 presents the result of such algorithm for different values of q. The case $q = 0$ is not represented as it corresponds to the classic non-robust set inversion with contractors presented in fig. 2.11(b). With $q = 1$, the algorithm display the sub-paving that only satisfies $m - q = 3$ observations, which corresponds to the solution we previously had without the faulty measurement. For $q = 2$, only 2 observations have to be satisfied, but the localization with only 2 distance measurements is ambiguous as two disjoint sub-pavings (symmetrical to the axis joining the 2 robots) can contain the solution. They are presented in fig. 2.10(b). Therefore the 2-relaxed intersection presented in fig. 2.12(b) is equivalent to the union of all pairs of distance measurements between two robots. Finally when $q = 3$, $m - q = 1$, which is basically the union of all the sets that satisfy one distance measurement.

2.4.5 GOMNE

The robust set inversion method presented in the previous subsection uses the q-intersection to compute sub-pavings guaranteed to contain the solution as long as the number of outliers is inferior or equal to a given q. In practice, q can be determined empirically for a specific problem. However, when q is overestimated, the

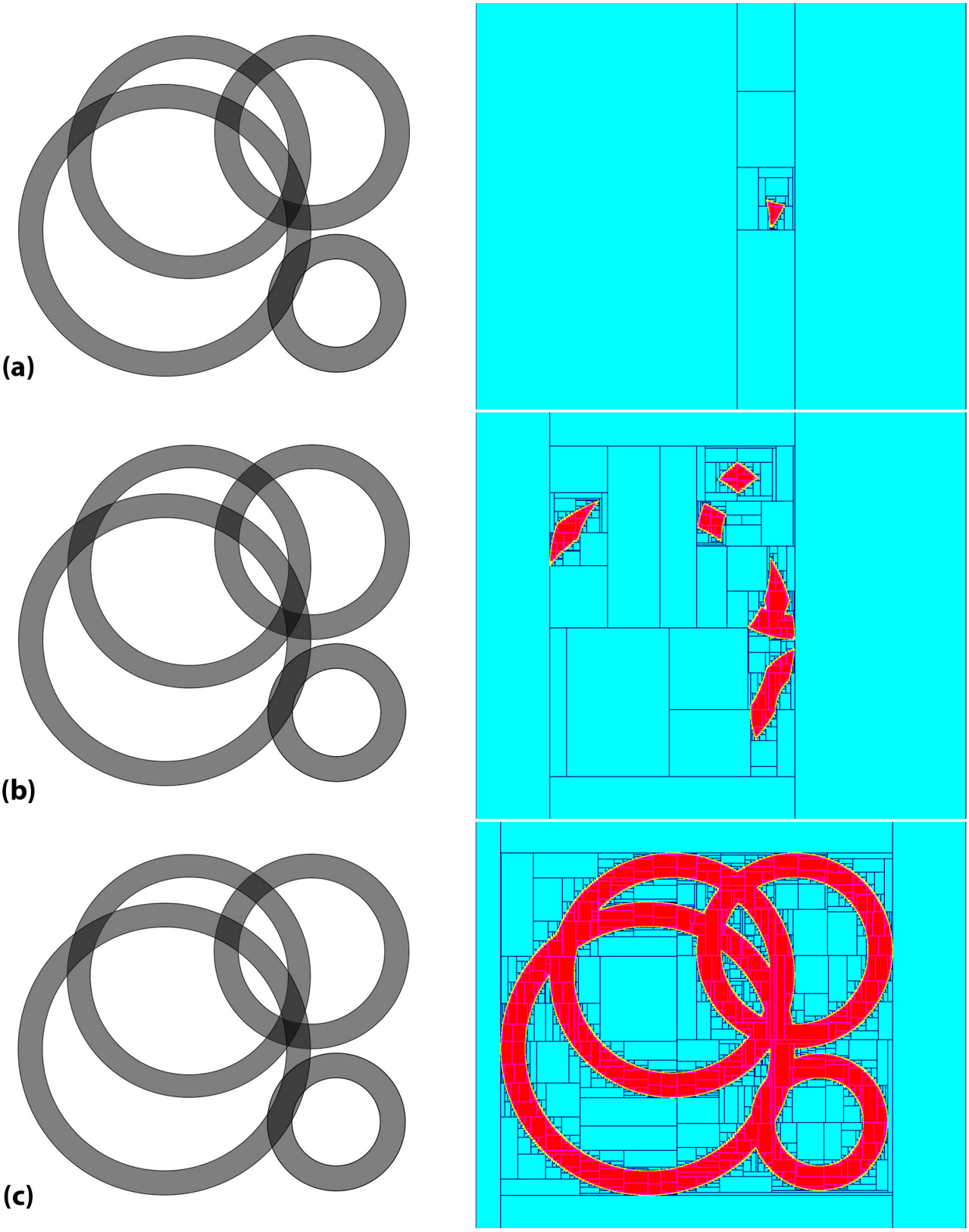

Figure 2.12: (a) 1-relaxed intersection, (b) 2-relaxed intersection and (c) 3-relaxed intersection.

	Algorithm $RSIVIA$ (in : $[\mathbf{x}], \mathbf{f}, q, [\mathbf{y}], \varepsilon$; inout: $\underline{\mathbb{X}}, \bar{\mathbb{X}}$)
1	$incl := 0; sep := 0;$
2	for $i = 1...m$ do
3	if $[f_i]([x]) \subset [y_i]$ then
4	$incl := incl + 1$
5	else if $[f_i]([x]) \cap [y_i] = \emptyset$
6	$sep := sep + 1$
7	end if
8	end for
9	if $incl \geq m - q$ then
10	$\underline{\mathbb{X}} = \underline{\mathbb{X}} \cup [\mathbf{x}]; \bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw([**x**], 'red'); return
11	else if $sep > q$ then
12	draw([**x**], 'blue'); return
13	else if $w([\mathbf{x}]) < \varepsilon$ then
14	$\bar{\mathbb{X}} = \bar{\mathbb{X}} \cup [\mathbf{x}]$;draw ([**x**], 'yellow'); return
15	else
16	bisect [**x**] into $[\mathbf{x}_1]$ and $[\mathbf{x}_2]$
17	$RSIVIA([\mathbf{x}_1], \mathbf{f}, q, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$
18	$RSIVIA([\mathbf{x}_2], \mathbf{f}, q, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$
19	end if

Table 2.4: SIVIA

solution set can be very pessimistic. A way to determine q automatically is to successively run $RSIVIA$ with an increasing q until the solution sub-paving is non-empty. This algorithm is called $GOMNE$ for *Guaranteed Outlier Minimal Number Estimator* [Jaulin et al., 1996] and is presented in table 2.5.

Running $GOMNE$ on our localization problem with an outlier, we obtain the same result as presented in fig. 2.12(a). The number of faulty measurements is correctly estimated by the algorithm. However the algorithm stops when the sub-paving $\underline{\mathbb{X}}$ is simply non-empty. This alone does not guarantee that the sub-paving encloses the solution. Indeed, some outliers might be consistent with each other and form a "false solution" on their own if the redundancy for the "real solution" is insufficient. Therefore, it is important to understand that $GOMNE$ computes the minimum number q_{min} of outliers with no guarantee on the real number. In order to increase the robustness of the algorithm, we can define a security margin r of

	Algorithm $GOMNE$ (in : $[\mathbf{x}], \mathbf{f}, [\mathbf{y}], \varepsilon$; inout: $q, \underline{\mathbb{X}}, \bar{\mathbb{X}}$)
1	$q := -1$
2	repeat
3	$\underline{\mathbb{X}} := \emptyset; \bar{\mathbb{X}} := \emptyset$
4	$q := q + 1$
5	$RSIVIA([\mathbf{x}_1], \mathbf{f}, q, [\mathbf{y}], \varepsilon, \underline{\mathbb{X}}, \bar{\mathbb{X}})$
6	until $\underline{\mathbb{X}} \neq \emptyset$

Table 2.5: SIVIA

non-detected outliers and actually compute the $(q_{min} + r)$-relaxed set inversion.

2.5 Conclusion

In this chapter, we presented the basic notions of interval arithmetic and a few algorithms to estimate parameters of a system given observations of this system with bounded errors. Using set inversion via interval analysis, we were able to compute an inner and outer approximation of a simple localization problem with an outlier.

In the next chapter we see how to extend contractor algebra to allow geometrical transformations of contractors.

The algorithms and notions presented in this chapter represents the state of the art in interval analysis. All copyrights go to cited authors.

Chapter 3

Computing optimal contractors using geometrical transformation

3.1 Introduction

In this chapter, we present the first contribution of this thesis and show that it is possible to extend contractor algebra to allow for the geometrical transformation of contractors and build optimal contractors in a very easy way for some constraints with symmetries. Many problems involve constraints with symmetries, i.e. central symmetry for the sine function, axial symmetry for the cosine function and 2π-modulation for both. We show that this extension makes it possible to build minimal contractors in an easy way for some constraints with symmetries. As an application, we consider the construction of the minimal contractor associated with the constraint $\theta = \text{atan2}(x, y)$ and demonstrate its application to robot localization.

This chapter is organized as follows : Section 2 reviews the notion of contractors and section 3 proposes new theorems for the application of symmetries to contractors. Section 4 proposes a new minimal contractor for atan2 based on symmetries, and section 5 presents an application to localize a robot in a group. Finally, the section 6 concludes the chapter.

3.2 Contractors

As a quick reminder, an *interval* of $\mathbb{R}$ is a closed connected subset of $\mathbb{R}$. A box $[\mathbf{x}]$ of $\mathbb{R}^n$ is the Cartesian product of n intervals. The set of all boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$. A *contractor* $\mathcal{C}$ is an operator $\mathbb{IR}^n \mapsto \mathbb{IR}^n$ such that

$$\begin{array}{ll} \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}] & \text{(contractance)} \\ [\mathbf{x}] \subset [\mathbf{y}] \;\Rightarrow\; \mathcal{C}([\mathbf{x}]) \subset \mathcal{C}([\mathbf{y}]) & \text{(monotonicity)} \end{array} \tag{3.1}$$

We define the inclusion between two contractors $\mathcal{C}_1$ and $\mathcal{C}_2$ as follows

$$\mathcal{C}_1 \subset \mathcal{C}_2 \Leftrightarrow \forall\, [\mathbf{x}] \in \mathbb{IR}^n,\, \mathcal{C}_1([\mathbf{x}]) \subset \mathcal{C}_2([\mathbf{x}]) \tag{3.2}$$

A set $\mathbb{S}$ is *consistent* with the contractor $\mathcal{C}$ (we will write $\mathbb{S} \sim \mathcal{C}$) if for all $[\mathbf{x}]$, we have

$$\mathcal{C}([\mathbf{x}]) \cap \mathbb{S} = [\mathbf{x}] \cap \mathbb{S}. \tag{3.3}$$

Two contractors $\mathcal{C}$ and $\mathcal{C}_1$ are *consistent* each other (we will write $\mathcal{C} \sim \mathcal{C}_1$) if for any set $\mathbb{S}$, we have

$$\mathbb{S} \sim \mathcal{C} \Leftrightarrow \mathbb{S} \sim \mathcal{C}_1. \tag{3.4}$$

A contractor $\mathcal{C}$ is *minimal* if for any other contractor $\mathcal{C}_1$, we have the following implication

$$\mathcal{C} \sim \mathcal{C}_1 \Rightarrow \mathcal{C} \subset \mathcal{C}_1. \tag{3.5}$$

A contractor represents a set of $\mathbb{R}^n$. The set associated with a contractor $\mathcal{C}$ is

$$\text{set}\,(\mathcal{C}) = \{\mathbf{x} \in \mathbb{R}^n, \mathcal{C}(\{\mathbf{x}\}) = \{\mathbf{x}\}\} \tag{3.6}$$

For instance, the set associated with the contractor

$$\mathcal{C}_1 \begin{pmatrix} [x_1] \\ [x_2] \\ [x_3] \end{pmatrix} \overset{\text{def}}{=} \begin{pmatrix} [x_1] \cap ([x_3] - [x_2]) \\ [x_2] \cap ([x_3] - [x_1]) \\ [x_3] \cap ([x_1] + [x_2]) \end{pmatrix} \tag{3.7}$$

is

$$\text{set}\,(\mathcal{C}_1) = \{(x_1, x_2, x_3)\,, x_3 = x_1 + x_2\} \tag{3.8}$$

A contractor is also one way to represent one equation $x_3 = x_1 + x_2$.

A set $\mathbb{S} \subset \mathbb{R}^n$ is said to be *functional* if there exists n monotonic functions $\varphi_1, \ldots \varphi_n : \mathbb{R}^{n-1} \to \mathbb{R}$ such that

$$\begin{aligned} (x_1, \ldots, x_n) \in \mathbb{S} \quad &\Leftrightarrow \quad x_1 = \varphi_1\,(x_2, \ldots, x_n) && (3.9)\\ &\Leftrightarrow \quad x_2 = \varphi_2\,(x_1, x_3, \ldots, x_n) && (3.10)\\ &\Leftrightarrow \quad \ldots && (3.11)\\ &\Leftrightarrow \quad x_n = \varphi_n\,(x_1, \ldots, x_{n-1}) && (3.12) \end{aligned}$$

For such a set, a minimal contractor $\mathcal{C}\,([\mathbf{x}])$ can easily be built by testing all end points of $[\mathbf{x}]$.

3.3 Extending contractor algebra for geometrical transformation

We define the following operations

intersection	$(\mathcal{C}_1 \cap \mathcal{C}_2)\,([\mathbf{x}]) \overset{\text{def}}{=} \mathcal{C}_1\,([\mathbf{x}]) \cap \mathcal{C}_2\,([\mathbf{x}])$
union	$(\mathcal{C}_1 \cup \mathcal{C}_2)\,([\mathbf{x}]) \overset{\text{def}}{=} [\mathcal{C}_1\,([\mathbf{x}]) \cup \mathcal{C}_2\,([\mathbf{x}])]$
composition	$(\mathcal{C}_1 \circ \mathcal{C}_2)\,([\mathbf{x}]) \overset{\text{def}}{=} \mathcal{C}_1\,(\mathcal{C}_2\,([\mathbf{x}]))$
repetition	$\mathcal{C}^\infty \overset{\text{def}}{=} \mathcal{C} \circ \mathcal{C} \circ \mathcal{C} \circ \ldots$
modulo	$(\mathcal{C} \bmod \mathbf{u})\,([\mathbf{x}]) = \sqcup_i \mathcal{T}_{i.\mathbf{u}} \circ \mathcal{C}\,([\mathbf{x}])$

where $\mathcal{T}$ is any transformation and $\mathcal{T}_{i.\mathbf{u}}$ is the translation of $i.\mathbf{u}$.

A function is said to be *box conservative* if

$$\mathbf{f}([\mathcal{X}]) = [\mathbf{f}(\mathcal{X})] \tag{3.13}$$

where $\mathcal{X}$ is a set of $\mathbb{R}^n$, $\mathbf{f}$ is a bijective function and $[\mathcal{X}]$ is the interval envelope of $\mathcal{X}$. This means that $\mathbf{f}^{-1}$ is also box conservative. Indeed,

$$\begin{aligned}
& \mathbf{f}^{-1}([\mathcal{X}]) = [\mathbf{f}^{-1}(\mathcal{X})] \\
\Leftrightarrow\ & \mathbf{f} \circ \mathbf{f}^{-1}([\mathcal{X}]) = \mathbf{f} \circ ([\mathbf{f}^{-1}(\mathcal{X})]) \text{ as } \mathbf{f} \text{ is bijective} \\
\Leftrightarrow\ & \mathbf{f} \circ \mathbf{f}^{-1}([\mathcal{X}]) = [\mathbf{f} \circ (\mathbf{f}^{-1}([\mathcal{X}]))] \text{ as } \mathbf{f} \text{ is box conservative} \\
\Leftrightarrow\ & [\mathcal{X}] = [\mathcal{X}] \text{ as } \mathbf{f} \circ \mathbf{f}^{-1} = \mathbf{Id}.
\end{aligned}$$

Therefore, $\mathbf{f}^{-1}([\mathcal{X}]) = [\mathbf{f}^{-1}(\mathcal{X})]$ and so $\mathbf{f}^{-1}$ is also box conservative.

Theorem. If $\mathcal{C}_{\mathbb{Y}}$ is a minimal contractor associated to $\mathbb{Y}$, and if $\mathbf{f}$ is box conservative (symmetry, translation, etc.) then $\mathbf{f}^{-1} \circ \mathcal{C}_{\mathbb{Y}} \circ \mathbf{f}$ is the minimal contractor associated to $\mathbb{X} = f^{-1}(\mathbb{Y})$.

Proof. The minimal contractor associated with $\mathbb{X}$ is

$$\begin{aligned}
\mathcal{C}_{\mathbb{X}}([\mathbf{x}]) &= [[\mathbf{x}] \cap \mathbb{X}] \\
&= [\mathbf{f}^{-1} \circ \mathbf{f}([\mathbf{x}]) \cap \mathbf{f}^{-1} \circ \mathbf{f}(\mathbb{X})] \text{ as } \mathbf{f}^{-1} \circ \mathbf{f} = \mathbf{Id} \\
&= [\mathbf{f}^{-1}(\mathbf{f}([\mathbf{x}]) \cap \mathbf{f}(\mathbb{X}))] \\
&= \mathbf{f}^{-1}([(\mathbf{f}([\mathbf{x}]) \cap \mathbf{f}(\mathbb{X}))]) \text{ as } \mathbf{f}^{-1} \text{ is box conservative} \\
&= \mathbf{f}^{-1}([\mathbf{f}([\mathbf{x}]) \cap \mathbb{Y}]) \text{ as } \mathbb{Y} = \mathbf{f}(\mathbb{X}) \\
&= \mathbf{f}^{-1}(\mathcal{C}_{\mathbb{Y}}(\mathbf{f}([\mathbf{x}]))) \text{ as } \mathbf{f}([\mathbf{x}]) \text{ is a box and } \mathcal{C}_{\mathbb{Y}}([\mathbf{y}]) = [[\mathbf{y}] \cap \mathbb{Y}] \\
&= \mathbf{f}^{-1} \circ \mathcal{C}_{\mathbb{Y}} \circ \mathbf{f}([\mathbf{x}]). \blacksquare
\end{aligned}$$

3.4 Computing a minimal contractor for Atan2

Atan2 is a function which is non monotonic and with discontinuities. A minimal contractor for the constraint $\blacksquare = atan2(y, x)$ cannot be obtained without decomposing the constraints on the plane (x, y) into boxes where the fonction is monototic. This is why we'll first apply the contractor to a monotonic part of the function then contract all other parts only using symmetries. The principle of using symmetries to build minimal contractors was proposed by *Pau Herrero* in his thesis [Herrero, 2006]. We define the constraint

$$\theta = \operatorname{atan2}(x, y) \Leftrightarrow \exists \ell \geq 0. \begin{cases} x = \ell \cos\theta \\ y = \ell \sin\theta \\ x^2 + y^2 = \ell^2 \end{cases} \tag{3.14}$$

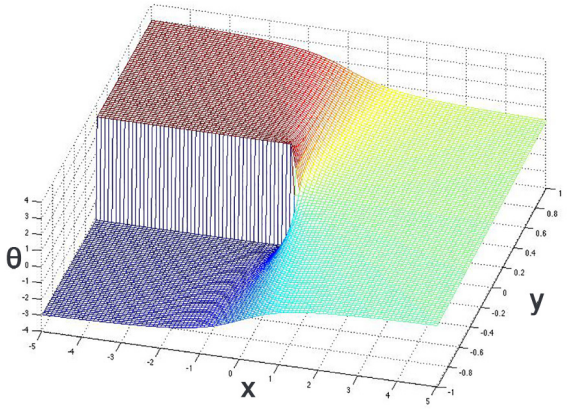

Figure 3.1: θ =atan2(y, x) displayed in MATLAB.

Fig. 3.1 presents the atan2 function.

The set associated with this constraint is denoted by $\mathbb{S}$, i.e.,

$$\mathbb{S} = \left\{(x, y, \theta) \in \mathbb{R}^3 \mid \theta = \text{atan2}\,(x, y)\right\} \tag{3.15}$$

We want to build the minimal contractor for this constraint.

3.4.1 Step 1 : Building the contractor on a monotonic box.

We define the set

$$\mathbb{S}_0 = \mathbb{S} \cap \left(\mathbb{R}^+ \times \mathbb{R}^+ \times [0, \frac{\pi}{2}]\right) \tag{3.16}$$

Since

$$(x, y, \theta) \in \mathbb{S} \Leftrightarrow \left\{\begin{array}{c} x = y.\text{cotan}\ \theta \\ y > 0, \\ \theta \in [0, \frac{\pi}{2}[\end{array}\right. \Leftrightarrow \left\{\begin{array}{c} y = x \tan \theta \\ x > 0, \\ \theta \in \left[0, \frac{\pi}{2}\right] \end{array}\right. \Leftrightarrow \left\{\begin{array}{c} \theta = \text{atan}\left(\frac{y}{x}\right) \\ x > 0, \\ y > 0. \end{array}\right. \tag{3.17}$$

the set $\mathbb{S}_0$ is functional with

$$\begin{array}{rcl} \varphi_x\,(y, \theta) & = & y.\text{cotan}\ \theta \text{ if } y > 0, \theta \in [0, \frac{\pi}{2}[\text{ and undefined otherwise} \\ \varphi_y\,(x, \theta) & = & x \tan \theta \text{ if } x > 0, \theta \in [0, \frac{\pi}{2}] \text{ and undefined otherwise} \\ \varphi_\theta\,(x, y) & = & \text{atan}\left(\frac{y}{x}\right) \text{ if } x > 0, y > 0 \text{ and undefined otherwise} \end{array} \tag{3.18}$$

The minimal contractor $\mathcal{C}_0$ is thus given by

$$\left(\begin{array}{c} [x] \\ [y] \\ [\theta] \end{array}\right) \rightarrow \left(\begin{array}{c} ([x] \cap \mathbb{R}^+) \cap ([y] \cap \mathbb{R}^+) \cdot \text{cotan}\ \left([\theta] \cap \left[0, \frac{\pi}{2}\right]\right) \\ ([y] \cap \mathbb{R}^+) \cap ([x] \cap \mathbb{R}^+) \cdot \tan\ \left([\theta] \cap \left[0, \frac{\pi}{2}\right]\right) \\ \left([\theta] \cap \left[0, \frac{\pi}{2}\right]\right) \cap \text{atan}\ \left(\frac{[y] \cap \mathbb{R}^+}{[x] \cap \mathbb{R}^+}\right) \end{array}\right) \tag{3.19}$$

Fig.3.2 shows the result when this contractor is applied in the $CSIVIA$ algorithm presented in the first chapter. The red sub-paving represents the inner approximation for x and y that verify $[\theta]$ =atan2$([x], [y])$. The union of the red and yellow sub-paving represents the outer approximation.

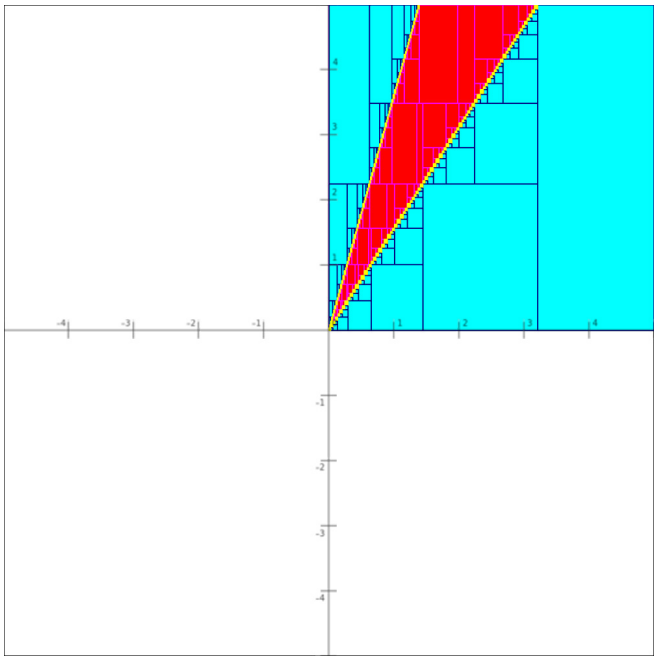

Figure 3.2: $CSIVIA$ applied to $\theta = \text{atan2}(y, x)$ for $[\theta] = [1, 1.3]$

3.4.2 Step 2 : Ox symmetry

Define the set

$$\mathbb{S}_1 = \mathbb{S} \cap \left(\mathbb{R}^+ \times \mathbb{R} \times [-\frac{\pi}{2}, \frac{\pi}{2}]\right) \tag{3.20}$$

We have

$$\mathbb{S}_1 = \mathbb{S}_0 \cup \mathcal{S}_{Ox}(\mathbb{S}_0) \tag{3.21}$$

where $\mathcal{S}_{Ox}$ is the symmetry with respect to the axis $0x$, defined by

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \rightarrow \begin{pmatrix} x \\ -y \\ -\theta \end{pmatrix} \tag{3.22}$$

Thus the minimal contractor associated to $\mathbb{S}_1$ is

$$\mathcal{C}_1 = \mathcal{C}_0 \cup \mathcal{S}_{Ox}(\mathcal{C}_0) \tag{3.23}$$

Numerically, the contractor $\mathcal{C}_1$ is built by applying $\mathcal{C}_0$ to $\text{atan2}(x, -y)$, then applying an Ox axial symmetry to the result. Fig.3.3 shows the result of this contractor on a set inversion via interval analysis.

3.4.3 Step 3 : Oy symmetry

Define the set

$$\mathbb{S}_2 = \mathbb{S} \cap (\mathbb{R} \times \mathbb{R} \times [-\pi, \pi]) \tag{3.24}$$

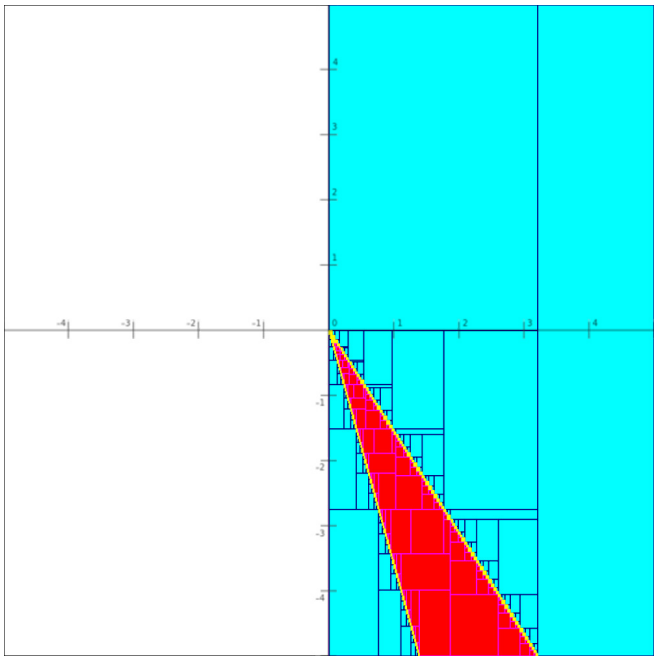

Figure 3.3: $CSIVIA$ applied to $\theta = \text{atan2}(y, x)$ for $[\theta] = [-1, -1.3]$

We have

$$\mathbb{S}_2 = \mathbb{S}_1 \cup \mathcal{S}_{Oy}(\mathbb{S}_0) \tag{3.25}$$

where $\mathcal{S}_{Oy}$ is the symmetry defined by

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \rightarrow \begin{pmatrix} -x \\ y \\ \pi - \theta \end{pmatrix} \tag{3.26}$$

Thus the minimal contractor associated to $\mathbb{S}_1$ is

$$\mathcal{C}_2 = \mathcal{C}_1 \cup \mathcal{S}_{Oy}(\mathcal{C}_1) \tag{3.27}$$

Numerically, the contractor $\mathcal{C}_2$ is built by applying $\mathcal{C}_1$ to $\text{atan2}(-x, y)$, then applying an Oy axial symmetry to the result. Fig.3.4 shows the result of this contractor on a set inversion via interval analysis.

3.4.4 Step 4 : 2π-modulo symmetry

An minimal contractor for $\mathbb{S}$ is

$$\mathcal{C} = (\mathcal{C}_2 \bmod \mathbf{u}) \tag{3.28}$$

where

$$\mathbf{u} = \begin{pmatrix} 0 \\ 0 \\ 2\pi \end{pmatrix} \tag{3.29}$$

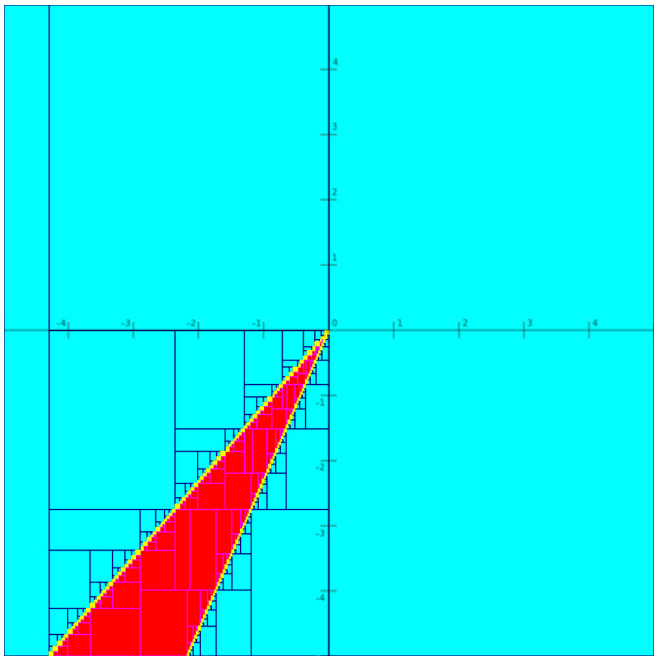

Figure 3.4: $CSIVIA$ applied to θ =atan2(y, x) for $[\theta] = [-2, -2.3]$

Numerically, the $\mathcal{C}$ contractor is built by applying a 2π modulation to θ then get it back between $[-\pi, \pi]$, and then applying the $\mathcal{C}_2$ contractor. the Fig.3.5 shows the result of this contractor on a set inversion via interval analysis. Notice that we are therefore not limited to $[-\pi, \pi]$, but can compute the set inversion for all $[\theta] \in \mathbb{R}$.

The final algorithm is presented in the table 3.1.

Notice that all the boxes of the sub-paving touch the yellow border, meaning that no bisection occurred and hence that the contractor proposed for atan2 is minimal.

3.5 Application to robot localization

Atan2 is used in many applications. One of them is bearings-only robot localization [Bishop et al., 2009] [Oshman and Davidson, 1999][Bekris et al., 2006][Logothetis et al., 1997]. Let's consider a group of n robots that can measure their angle in respect to each other. The feasible set for the position of the robots is:

$$\mathbb{X} = \{(x_i, y_i) | \forall i, j \in \{1, 2, .., n\}, i \neq j, \theta_{ij} = \text{atan2}(x_i - x_j, y_i - y_j)\} \tag{3.30}$$

where (x_i, y_i) is the position of the robot i, and θ_{ij} is the measured angle between this robot and another robot j. We want to find an inner and an outer approximation of the feasible set.

Let's consider $n = 4$ robots. The positions of robots 1, 2 and 3 are known but the position of robot 4 is unknown. However it can measure:

	Algorithm C_{ATAN2} (inout : $[x], [y], [\theta]$)
1	if $[x]$ in $\mathbb{R}^+$ and $[y]$ in $\mathbb{R}^+$ and $[\theta]$ in $[0, \frac{\pi}{2}]$
2	$\begin{pmatrix} [x] \\ [y] \\ [\theta] \end{pmatrix} = \begin{pmatrix} \text{cotan } ([\theta]) \\ \tan ([\theta]) \\ \text{atan } \left(\frac{[y]}{[x]}\right) \end{pmatrix}$
3	return $([x], [y], [\theta])$;
4	else
	$[\theta] = \text{mod}([\theta], 2\pi)$
5	$[x_1] = [x] \cap [0, +\infty)$
6	$[y_1] = [y] \cap [0, +\infty)$
7	$[\theta_1] = [\theta]$
8	C_{ATAN2} $([x_1], [y_1], [\theta_1])$
9	$[x_2] = [x] \cap [0, +\infty)$
10	$[y_2] = [y] \cap (-\infty, 0]$
11	$[\theta_2] = -[\theta]$
12	C_{ATAN2} $([x_2], [y_2], [\theta_2])$
13	$[x_3] = [x] \cap (-\infty, 0]$
14	$[y_3] = [y] \cap (-\infty, 0]$
15	$[\theta_3] = [\pi - \theta]$
16	C_{ATAN2} $([x_3], [y_3], [\theta_3])$
17	$[x_4] = [x] \cap (-\infty, 0]$
18	$[y_4] = [y] \cap [0, +\infty)$
19	$[\theta_4] = [\theta - \pi]$
20	C_{ATAN2} $([x_4], [y_4], [\theta_4])$
21	$[x] = [x_1] \cup [x_2] \cup (-[x_3]) \cup (-[x_4])$
22	$[y] = [y_1] \cup (-[y_2]) \cup (-[y_3]) \cup [y_4]$
23	$[\theta] = [\theta_1] \cap (-[\theta_2]) \cap (\pi - [\theta_3]) \cap (\pi + [\theta_4])$
29	return $([x], [y], [\theta])$;

Table 3.1: Atan2 Contractor

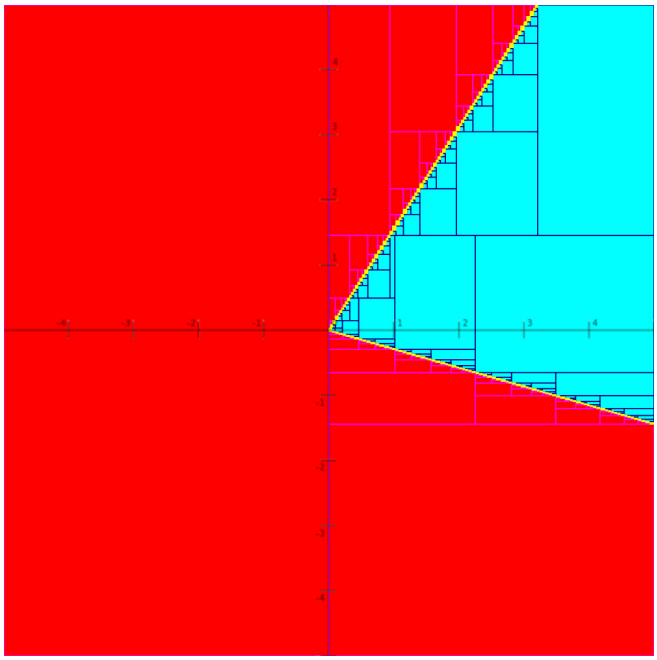

Figure 3.5: $CSIVIA$ applied to $\theta = \text{atan2}(y, x)$ for $[\theta] = [6, 7]$

· $[\theta_{41}] = [0.2, 0.3]$ to robot 1 at position $(-3, -3)$ presented in fig 3.6(a),

· $[\theta_{42}] = [5.55, 5.65]$ to robot 2 at position $(-4, 4)$ presented in fig 3.6(b),

· $[\theta_{43}] = [4.4, 4.5]$ to robot 3 at position $(3, 1)$ presented in fig 3.6(c).

The resulting sub-pavings of $CSIVIA$ enclose the position of robot 4.

As in the first chapter, we can apply a q-relaxed intersection if we suspect the presence of outliers (fig. 3.7).

3.6 Conclusion

This chapter extended contractor algebra to allow for the geometrical transformation of contractors and showed that it is possible to build minimal contractors in a very easy way for some constraints with symmetries. As an application, we considered the construction of a contractor associated with the constraint $\theta = \text{atan2}(x, y)$ using central, axial and 2π-modulo symmetries, and showed that this contractor was minimal. The test case has been realized using the *IBEX (Interval Based EXplorer)* library developed by *Gilles Chabert* (http://www.ibex-lib.org/). The contractor proposed for *atan2* has been approved for official release in the next version of *IBEX*. In the meantime, all source codes are available for download on http://aymericbethencourt.com/thesis.

So far we have only considered static problems in this thesis. However time is often a variable that we have to consider in real problems. In order to consider dynamic problems, we have to develop a new tool that will enclose intervals and boxes at different times.

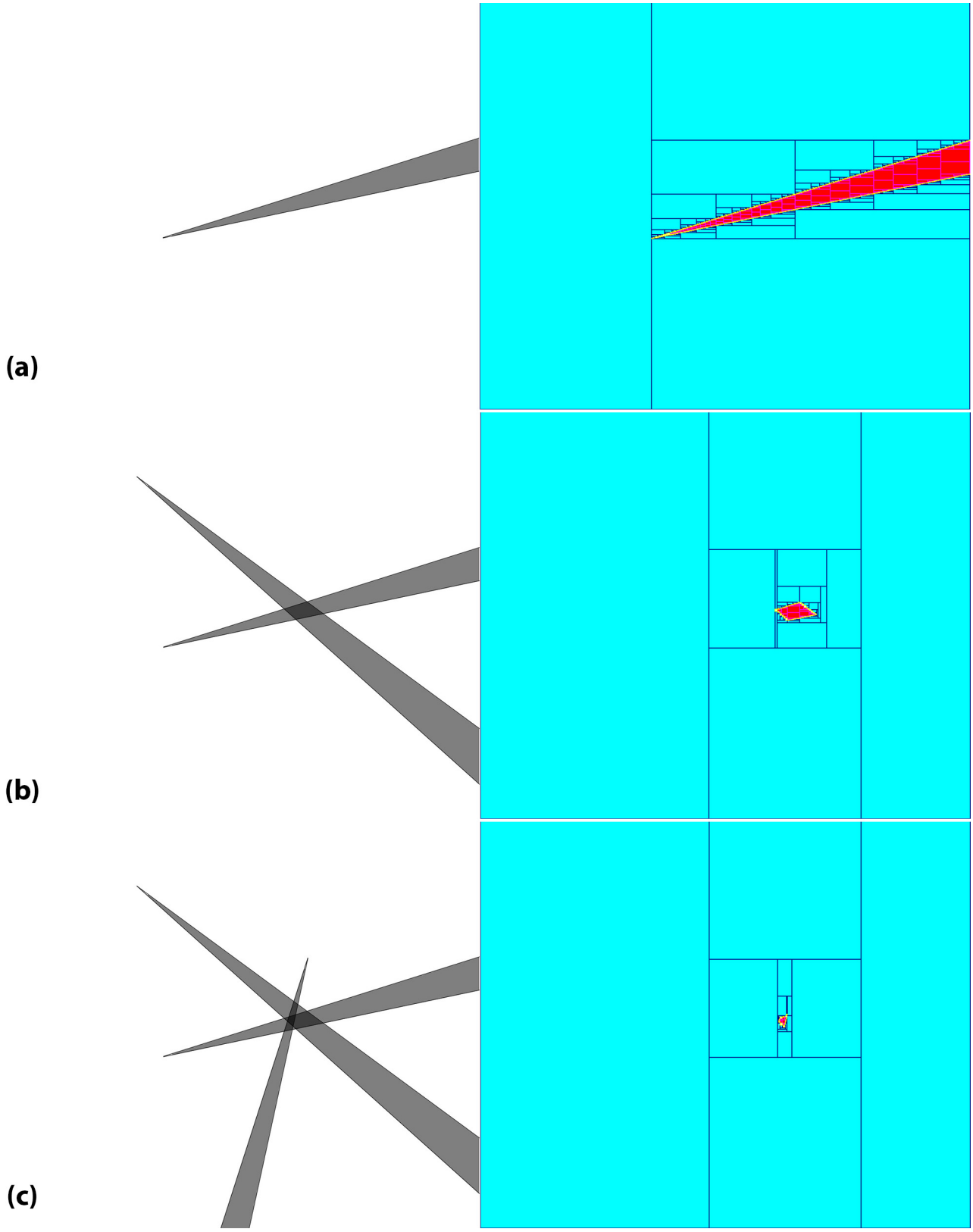

Figure 3.6: (a) Angle measurement to robot 1, (b) to robot 1 and 2 and (c) to robot 1,2 and 3.

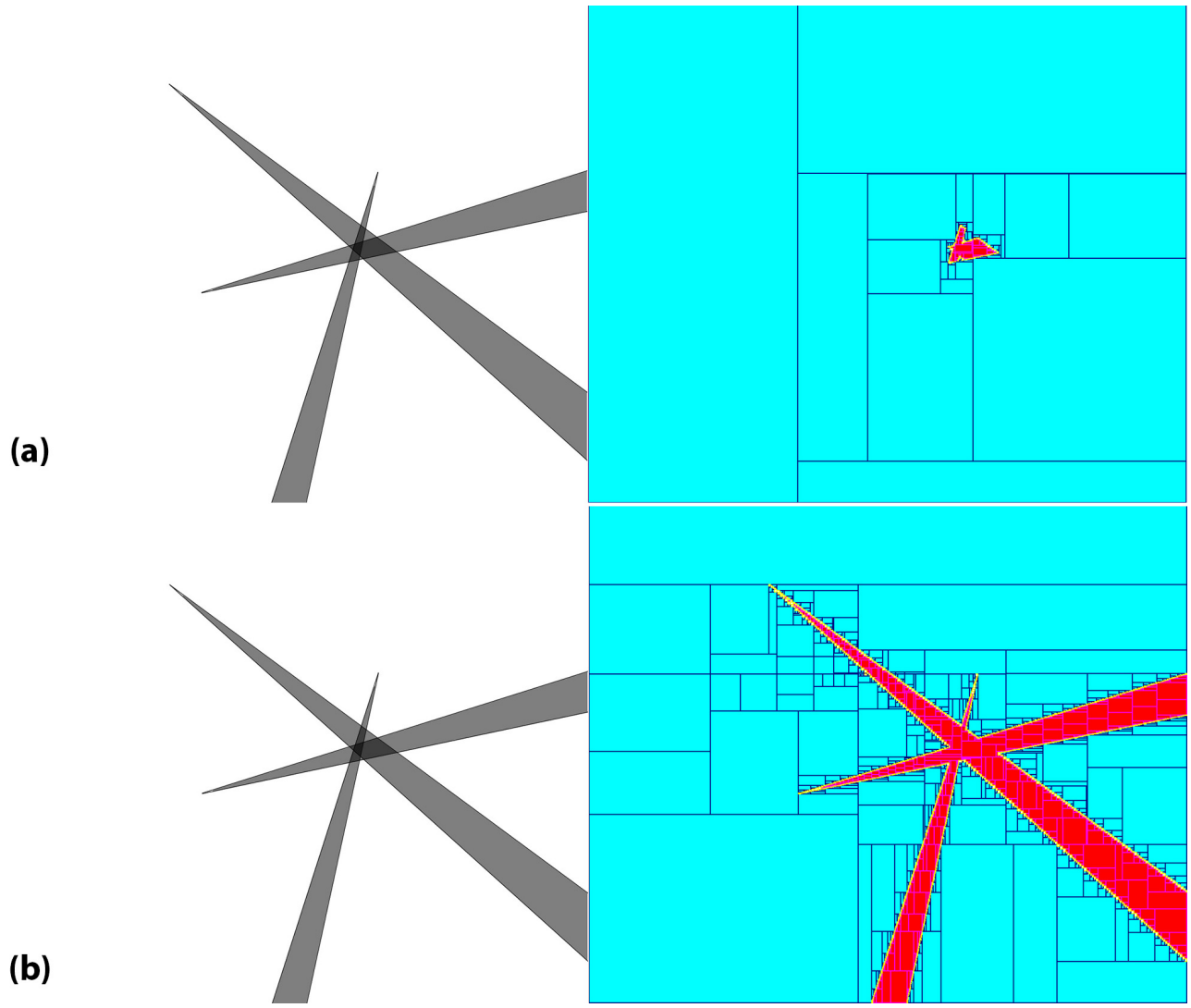

Figure 3.7: (a) 1-relaxed intersection and (b) 2-relaxed intersection.

The work presented in this chapter has been **submitted** to: *Reliable Computing* , 2014, and is currently awaiting reviews.

Chapter 4

Solving non-linear constraint satisfaction problems involving time-dependant functions

4.1 Introduction

In this chapter, we consider dynamic localization problems where time is an important variable. In other words, we look to resolve non-linear constraint satisfaction problems where the variables of the systems are trajectories (functions from $\mathbb{R}$ to $\mathbb{R}^n$) [Drevelle and Bonnifait, 2009]. We introduce the notion of *tubes* (or *intervals of functions*), inspired from Taylor models [Berz and Makino, 1998], and for which the lower and upper bounds are trajectories with respect to the inclusion. We then define basic operators and prove a few propositions verified by tubes in. We show the possibility to build contractors on tubes and propagate constraints to solve problems involving time-dependant functions as the unknown variables. We show that this approach can be particularly powerful when inter-temporal equations (e.g. delays) are involved. Finally, in order to illustrate the principle and efficiency of the approach, several test cases are provided.

4.2 Intervals of functions (or *tubes*)

In interval analysis, the unknown variables are usually boolean numbers, integers or real numbers, but the originality of this chapter is to consider trajectories.

4.2.1 Tubes

A *Tube* (or *interval of a trajectory*) [Kurzhanski and Valyi, 1997][Milanese et al., 1996] is a set-membership vision of a random signal. A *tube* $[\mathbf{x}]$ is an interval $[\mathbf{x}^-, \mathbf{x}^+]$ of $\mathcal{F}^n$, i.e., a pair of two trajectories $\mathbf{x}^-, \mathbf{x}^+$ such that for all t, $\mathbf{x}^-(t) \leq \mathbf{x}^+(t)$. The set of all tubes of $\mathcal{F}^n$ is denoted by $\mathbb{I}\mathcal{F}^n$. In the coninuous case, a tubes is also called an interval-valued function, a special case of set-valued functions.

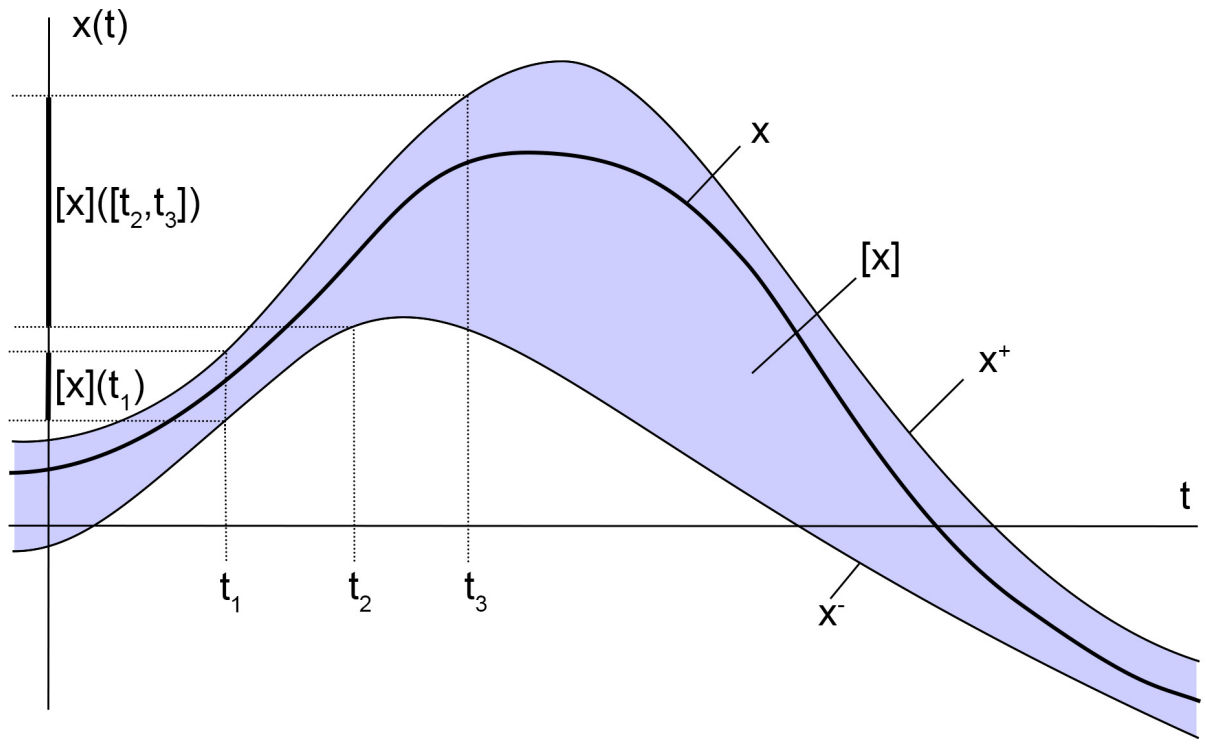

Figure 4.1: A tube $[x]$ of $\mathbb{R}$ which encloses the function x

An element $\mathbf{x}$ of $\mathcal{F}^n$ belongs to the tube $[\mathbf{x}]$ if $\forall t, \mathbf{x}(t) \in [\mathbf{x}](t)$. Fig. 4.1 illustrates a function $x \in \mathcal{F}^1$ which is inside a tube $[x]$. This tube gives us information related to the unknown function x.

If $\mathbf{x}$ is a function from $\mathbb{R}$ to $\mathbb{R}^n$ (i.e., $\mathbf{x} \in \mathcal{F}^n$), we define

$$\mathbf{x}([t]) = \{\mathbf{x}(t), t \in [t]\}. \tag{4.1}$$

Numerically, a tube $[\mathbf{x}]$ is defined by

$$[\mathbf{x}]([t]) = \bigsqcup_{t \in [t]} [\mathbf{x}](t), \tag{4.2}$$

i.e., $[\mathbf{x}]([t])$ is the smallest box which encloses all boxes $[\mathbf{x}](t), t \in [t]$. It is easy to prove that

$$\mathbf{x} \in [\mathbf{x}], t \in [t] \Rightarrow \mathbf{x}(t) \in [\mathbf{x}]([t]), \tag{4.3}$$

and that no box smaller than $[\mathbf{x}]([t])$ satisfies this property.

Given two trajectories x and y, the least upper bound of x and y according to a pointwise order is called the *join* or *supremum* and is denoted by $x \vee y$. The greatest lower bound according to a pointwise order is called the *meet* or *infimum* and is written as $x \wedge y$.

4.2.2 Tube arithmetic

We can extend operations on intervals to tubes. The extension is the smallest tube which encloses the solution. Therefore we can extend operations such as the sum, multiplication, image by a function, etc. to

tubes. We use the rules of interval arithmetic and inclusion functions [Moore, 1979]. An arithmetic on tubes is thus a direct extension of interval arithmetic. As it is the case for interval computation, the result of an operation on tubes contains all results of the same operation performed on the enclosed elements of $\mathcal{F}^n$.

Integral. Consider two numbers t_1, t_2 such that $t_2 \geq t_1 \geq 0$. The integral of a tube $[\mathbf{x}]$ over an interval $[t_1, t_2]$ is defined [Aubry et al., 2013] by

$$\int_{t_1}^{t_2} [\mathbf{x}](\tau)\, d\tau = \left\{ \int_{t_1}^{t_2} \mathbf{x}(\tau)\, d\tau \text{ such that } \mathbf{x} \in [\mathbf{x}] \right\}. \tag{4.4}$$

We deduce from the monotonicity of the integral operator that

$$\int_{t_1}^{t_2} [\mathbf{x}](\tau)\, d\tau = \left[\int_{t_1}^{t_2} \mathbf{x}^-(\tau)\, d\tau, \int_{t_1}^{t_2} \mathbf{x}^+(\tau)\, d\tau \right]. \tag{4.5}$$

where $[x] = [x^-, x^+]$. From the definition of tube integrals, we have

$$\mathbf{x} \in [\mathbf{x}] \Rightarrow \int_{t_1}^{t_2} \mathbf{x}(\tau)\, d\tau \in \int_{t_1}^{t_2} [\mathbf{x}](\tau)\, d\tau. \tag{4.6}$$

Moreover, the *interval primitive* defined by $\int_0^t [\mathbf{x}](\tau)\, d\tau$ defines a tube that vanishes for $t = 0$.

Extension of operators. If $\diamond$ is a binary operator in $\mathbb{R}^n$(such as $+$,$-$,the multiplication $*$ when $n = 1$ or the dot product when $n >= 2$) then it can be extended to $\mathcal{F}^n$ (in the Minkowski sense) and to $\mathbb{I}\mathcal{F}^n$ as follows

$$([x] \diamond [y])(t) = ([x](t) \diamond [y](t)) \tag{4.7}$$

E.g., for $[\mathbf{x}]$ and $[\mathbf{y}] \in \mathbb{I}\mathcal{F}^n$ and $a \in \mathbb{R}^+$

$$\begin{array}{cc} [z] = [x] + [y] \Longrightarrow \forall t, [z](t) = [x](t) + [y](t) & \text{(sum)} \\ [z] = shift_a([x]) \Longrightarrow \forall t, [z](t) = [x](t+a) & \text{(shift)} \\ [z] = [x] \circ [y] \Longrightarrow \forall t, [z](t) = [x]([y](t)) & \text{(composition)} \\ [z] = \int [x] \Longrightarrow [z](t) = \left[\int_0^t \mathbf{x}^-(\tau)\, d\tau, \int_0^t \mathbf{x}^+(\tau)\, d\tau \right] & \text{(integral)} \end{array} \tag{4.8}$$

Consider a collection $\{f_i, i \in \mathbb{N}\}$ of functions. The *tube envelope* $\square\{f_i, i \in \mathbb{N}\}$ is the smallest tube enclosing all f_i. We have

$$\square\{f_i, i \in \mathbb{N}\} = [\wedge_{i \in \mathbb{N}} f_i; \vee_{i \in \mathbb{N}} f_i)] \tag{4.9}$$

For instance,

$$\text{for } t \in [0, \infty], \square\{t, \cos(t), -1\} = [-1; t] \tag{4.10}$$

We define the *tube inclusion* as follows

$$[x] \sqsubset [y] \Longleftrightarrow y^- \leqslant x^- \leqslant x^+ \leqslant y^+ \tag{4.11}$$

We define the *tube intersection* as follows

$$[x] \sqcap [y] \Longleftrightarrow [[x] \vee [y], [x] \wedge [y]] \tag{4.12}$$

4.2.3 Constraint propagation on tubes

Consider a constraint $\mathcal{L}(x)$ on a trajectory x of $\mathcal{F}^n$. A contractor associated with the constraint $\mathcal{L}$ is an operator $C_{\mathcal{L}}$

$$\begin{aligned} C_{\mathcal{L}}: \quad \mathbb{I}\mathcal{F}^n &\longrightarrow \mathbb{I}\mathcal{F}^n \\ [y] &\longmapsto [x] \end{aligned}$$

where $[x]$ and $[y]$ are tubes, such that

$$\begin{array}{ll} \forall t, [x](t) \subset [y](t) & \text{(contraction)} \\ \left(\begin{array}{c} \mathcal{L}(x) \\ x \in [x] \end{array} \right) \Longrightarrow x \in [y] & \text{(completeness)} \end{array} \tag{4.13}$$

We call C^* the *minimal contractor* for $\mathcal{L}$ that returns the tube $[y]$ with the smallest width and is consistent with the constraint $\mathcal{L}$.

We can propose minimal contractors for tubes, but let us first separate our constraints into two categories:

• *Synchronous constraint*: Constraint that is set to happen at the same time, typically $x(t) = y(t)$ is synchronous.

• *Asynchronous constraint*: Constraint that is set to happen at different times, typically $x(t) = y(t-2)$ is asynchronous. This requires to have the history of the variable y and this is where tubes will be particularly powerful.

We are now proposing a few minimal contractors for both types.

Proposition 1. The minimal contractor associated with the synchronous constraint $x \leqslant y$ or $\forall t \in [0, \infty[, x(t) \leqslant y(t)$ is

$$C_{\leqslant} \left(\begin{array}{c} [x] \\ [y] \end{array} \right) = \left(\begin{array}{c} [x^-, x^+ \wedge y^+] \\ [x^- \vee y^-, y^+] \end{array} \right) \tag{4.14}$$

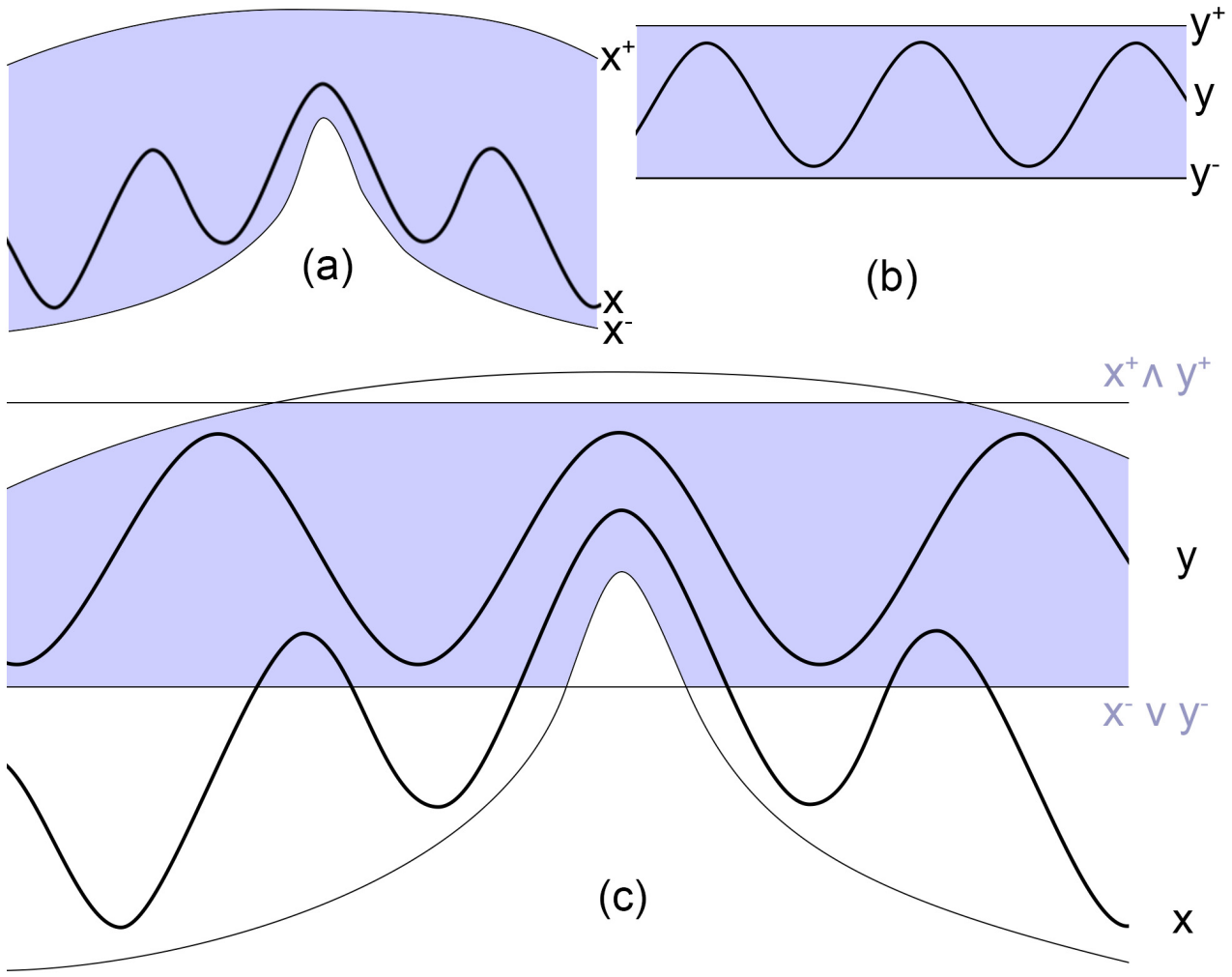

Figure 4.2: (a) represents the trajectory x and its tube $[x]$, (b) the trajectory y and its tube $[y]$, and (c) illustrates the supremum $x^+ \wedge y^+$ (upper bound of the purple area) and infimum $x^- \vee y^-$ (lower bound of purple area).

Proof. $x \leqslant y$ so we can find a such that $y = x + a$ with $a \geqslant 0$. We can therefore contract the associated tube.

$$\begin{aligned} [y] &= [y] \cap ([x] + [a]) \\ \iff [y] &= [y^- \vee (x^- + a^-), y^+ \wedge (x^+ + a^+)] \\ \iff [y] &= [y^- \vee (x^- + 0), y^+ \wedge (x^+ + \infty)] \\ \iff [y] &= [y^- \vee x^-, y^+] \end{aligned} \tag{4.15}$$

since $[a] = [0, \infty[$. By considering $x = y + a$ with $a \leqslant 0$, we can prove that $[x] = [x^-, x^+ \wedge y^+]$ the same way.■

Fig. 4.2 illustrates the contractions.

Proposition 2. The minimal contractor associated with the synchronous constraint $x = y$ (which means

that $\forall t \in [0,\infty], x(t) = y(t)$) is

$$C_{=}\begin{pmatrix}[x]\\[y]\end{pmatrix} = \begin{pmatrix}[x]\cap[y]\\[x]\cap[y]\end{pmatrix} \tag{4.16}$$

Proof. As $x = y$, any given elements of x and y cannot exceed each other's range. Both variables will share a new interval equal to the intersection of their respective intervals

$$\begin{aligned} & \forall t, [x](t) = [x](t)\cap[y](t) \\ \iff \quad & [x] = [x]\cap[y] \end{aligned} \tag{4.17}$$

The same is proven for y. Notice that we could also make the proof by aplying the superiority contractor to $x \leqslant y$ then $y \geqslant x$. ■

Proposition 3. The minimal contractor associated with a translation or delay $\forall t \in [0,\infty[, x(t) = y(t-\tau)$ is

$$C_{delay}\begin{pmatrix}[x]\\[y]\end{pmatrix} = \begin{pmatrix}[x]\cap[y](t-■)\\[y]\cap[x](t+■)\end{pmatrix} \tag{4.18}$$

Proof. The proof is immediate considering Proposition 2.

Fig. 4.3 illustrates this notion. The purple area represents $\forall t, [x](t)\cap[y](t-■)$

Let us notice that this constraint is asynchronous as it involves two different times: t and $t-\tau$.■

Special case. If the constraint is $x(t) = x(t-\tau)$, then x is τ-periodic and $\forall k \in \mathbb{N}, C_{periodic}([x]) = [x]\cap[x](t-■)\cap[x](t-2■)\cap ...\cap[x](t-k■)$

Proposition 4. The minimal contractor associated with an axial symmetry around the axis $t = \tau$ is

$$C_{ASym}([x(■+t)]) = [x](■+t)\cap[x](■-t) \tag{4.19}$$

Proof. Let's define the axial symmetry as an increasing operator S on trajectories such that $(S(f))(\tau+t) = f(\tau-t)$.

Moreover,

$$\begin{aligned} [x(\tau+t)] &= [x^-(\tau+t), x^+(\tau+t)] \\ \Leftrightarrow S([x(\tau+t)]) &= [S(x^-(\tau+t)), S(x^+(\tau+t))] \\ \Leftrightarrow S([x(\tau+t)]) &= [x^-(\tau-t), x^+(\tau-t)] \text{ as } S \text{ is increasing.} \end{aligned} \tag{4.20}$$

Therefore $C_{ASym}([x(\tau+t)]) = [x](■+t)\cap S([x(\tau+t)]) = [x](■+t)\cap[x](■-t)$. ■

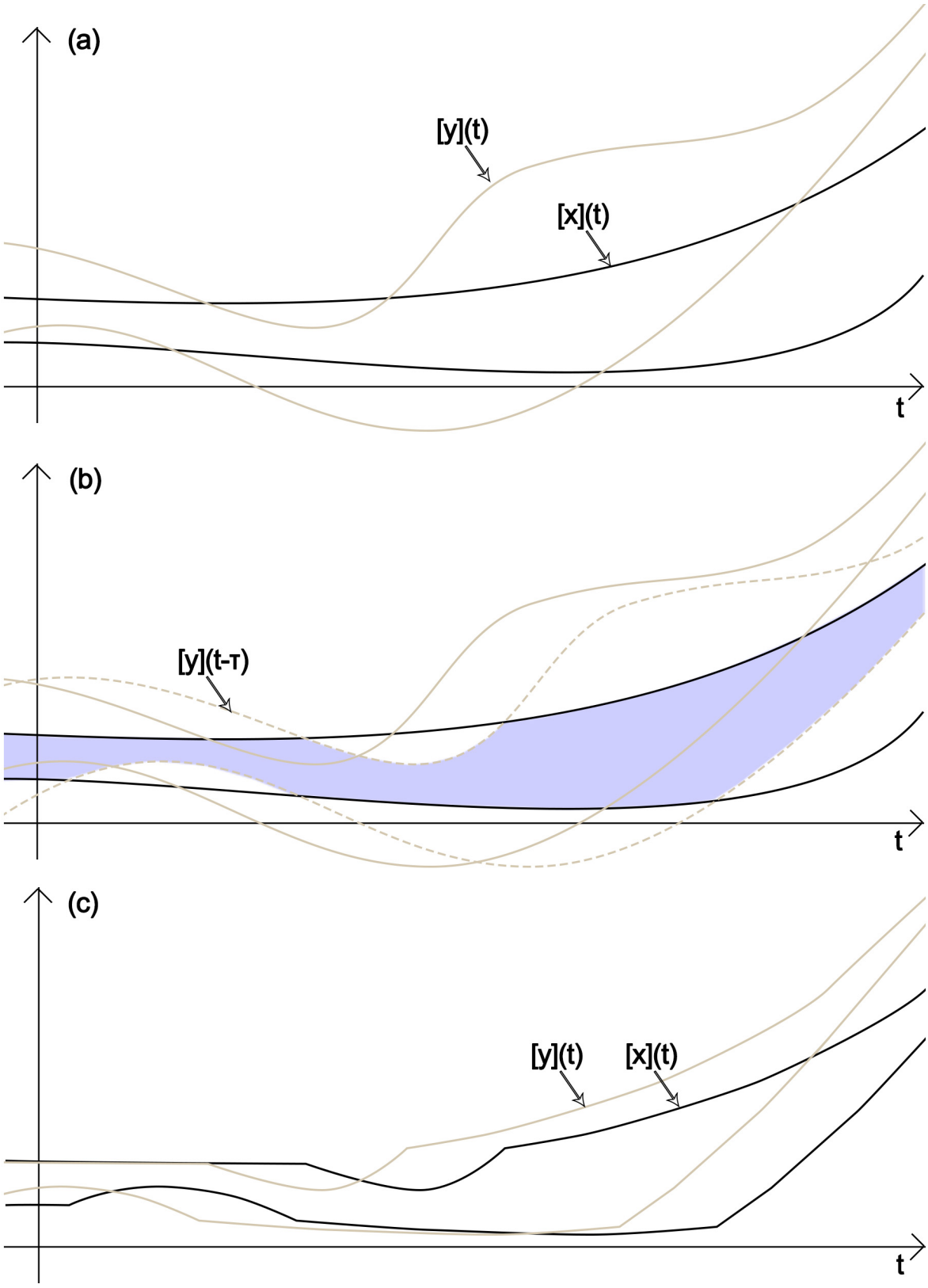

Figure 4.3: Asynchronous constraint propagation on tubes.

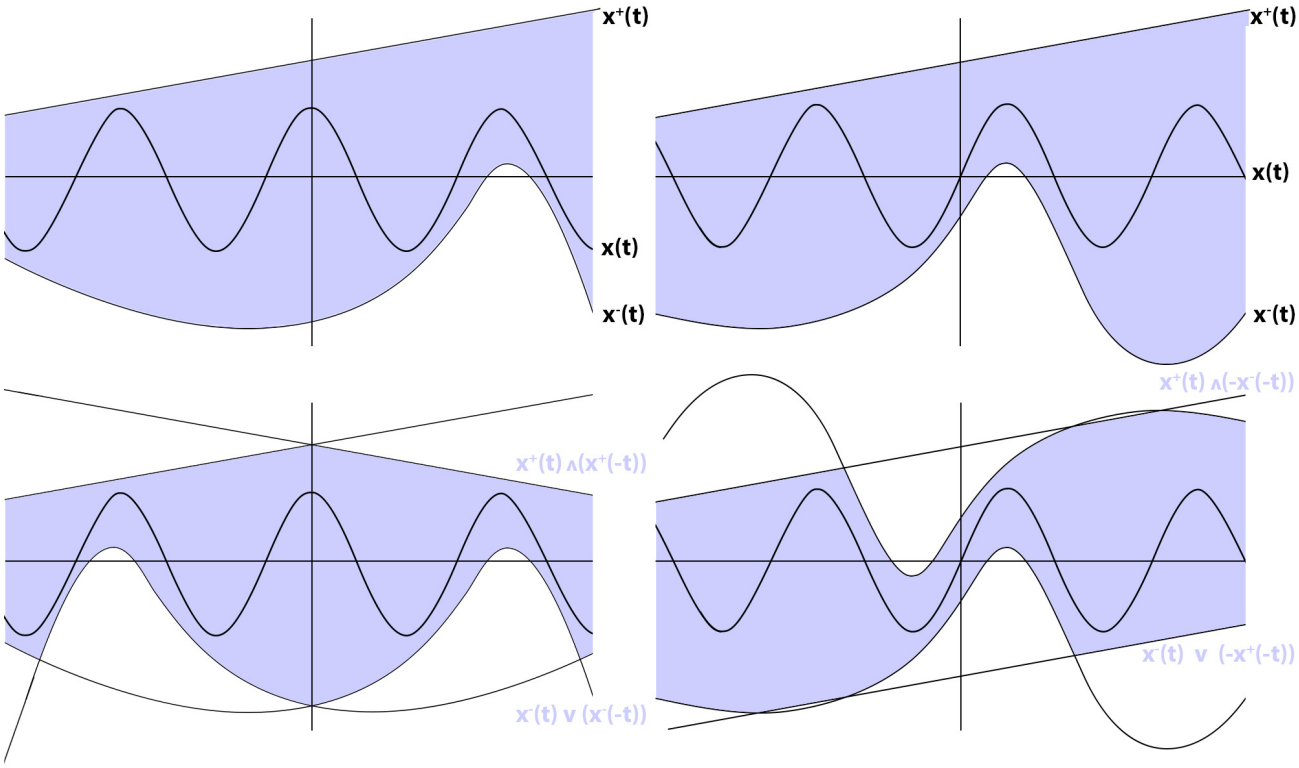

Figure 4.4: On the left, we illustrate the axial symmetry. On the right, the central symmetry.

Special case : When $\tau = 0$, then x is even and $C_{even}([x(t)]) = [x^-(t) \vee x^-(-t); x^+(t) \wedge x^+(-t)]$

Fig.4.4 illustrates this case.

Proposition 5. The minimal contractor associated with a central symmetry around the axis $t = \tau$ is

$$C_{CSym}([x(■ + t)]) = [x](■ + t) \cap -[x](■ - t) \tag{4.21}$$

Proof. The proof is identical to proposition 4 considering S decreasing.■

Special case : When $\tau = 0$, then x is odd and $C_{odd}([x(t)]) = [x^-(t) \vee (-x^+(-t)); x^+(t) \wedge (-x^-(-t))]$

Notice that both symmetries are asynchronous constraints.

4.2.4 State estimation

The problem to be considered in this section is the state estimation of a non-linear continuous-time systems in a bounded error context. Usually we describe the system as follows:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t)) \end{aligned} \tag{4.22}$$

where $t \in [0, \infty[$ is the time, $\mathbf{x}(t) \in \mathbb{R}^n$ and $\mathbf{y}(t) \in \mathbb{R}^n$ are respectively the state and the output vectors at time t, and $\mathbf{f}$ and $\mathbf{g}$ the evolution and observation functions. The particularity of our approach is to consider that $\mathbf{g}$ is no longer the observation function at a given state, but a relationship between two states of the system at different times. $\mathbf{g}$ is therefore an inter-temporal equation between the state of the system at time t_i and its state at time t_j. The system (4.22) becomes

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) \\ \mathbf{y}(t_i, t_j) &= \mathbf{g}(\mathbf{x}(t_i), \mathbf{x}(t_j)) \end{aligned} \tag{4.23}$$

This representation allows us to describe inter-temporal constraints, and use them to solve time-dependent problems as presented in the next section. An *Extended Kalman Filter* (EKF) [Ljung, 1979] could be considered, but the inter-temporality of the constrainsts make it hard to work with. Our membership approach is based on variants of the *CONTRACT* and *STRANGLE* algorithms from [Jaulin, 2002].

We propose the following way to estimate the smallest tube around the solution(s). We first start with a large tube enclosing $\mathbf{x}$ and contract $[\mathbf{x}]$ for all t using the output vectors, i.e. $\mathbf{y}(t_i, t_j)$ will contract $[\mathbf{x}](t_i)$ and $[\mathbf{x}](t_j)$. We then apply the *STRANGLE* algorithm from [Jaulin, 2002] presented in the table 4.1. We first apply the *STRANGLE* algorithm at a given time k_0 and initial box $[\mathbf{box}]$ with a precision of ϵ. $[\mathbf{box}]$ is the box we fix at time k_0, chosen initially big enough to make sure it contains the solution. We then apply all contractors and see if the resulting tubes are empty. If they are empty, $[\mathbf{box}]$ does not contain the solution at time k_0. If the tube is not empty, then $[\mathbf{box}]$ does not contain the solution at time k_0. We then bisects the box $[\mathbf{box}]$ into two sub-boxes and propagates each sub-box using the *CONTRACT* algorithm presented in the table 4.2. If the contracted tube using one of the sub-boxes possesses an empty interval at any t, then the sub-box does not contain the solution and we discard it. If the contracted tube has no empty interval, then the sub-box contains the solution and *STRANGLE* bisects it again and call *CONTRACT* recursively until the precision is reached. We then take the union of all non-empty tubes. The union is guaranteed to contain the solution, in regard, of course, of the chosen model and the integration method.

The *CONTRACT* algorithm uses the operators as defined in [Jaulin, 2002] to compute enclosures of the state vector at times $\delta, 2\delta, ...\bar{k}\delta$ where δ is the sampling time and $\bar{k}$ is the largest integer smaller than $\bar{t}/\delta$, from a given box $[\mathbf{x}]$ containing $\mathbf{x}$. $\bar{t}$ is the end time of the problem. $[\boldsymbol{\phi}]$ is the operator for the time increasing that computes the state of the system at time $k+1$ from k. $[\tilde{\boldsymbol{\phi}}]$ is the operator for the time decreasing that computes the state of the system at time k from $k+1$. For the sake of simplicity in the following examples, we are using a basic *Euler method* for $[\boldsymbol{\phi}]$ and thus for solving the ODE by computing the successive states of the system. Notice that using an *Euler method* means that we lose the interval guarantee but this does not impact the proof of concept. Moreover, there exist multiple libraries for computing rigorous bounds on the solution of ordinary differential equations, e.g. *VNODE* [Nedialkov, 2006] and methods especially designed for studying the evolution of dynamic systems [Bouissou et al., 2013].

	Algorithm $C_{CONTRACT}$ (in : $k_0, [\mathbf{box}]$, inout: $[\mathbf{x}]$)
1	for $k := 0$ to $\bar{k}$
2	if($k == k_0$)
3	$[\mathbf{x}](k) = [\mathbf{box}]$;
5	endif
6	$[\mathbf{x}](k+1) = [\mathbf{x}](k+1) \cap [\phi]([\mathbf{x}](k), [\dot{\mathbf{x}}](k))$;
13	endfor
14	for $k := \bar{k}$ to 0
15	if($k == k_0$)
16	$[\mathbf{x}](k) = [\mathbf{box}]$;
18	endif
19	$[\mathbf{x}](k) = [\mathbf{x}](k) \cap [\tilde{\phi}]([\mathbf{x}](k+1), [\dot{\mathbf{x}}](k+1))$;
20	endfor
21	return $[\mathbf{x}]$;

Table 4.1: Contract algorithm

	Algorithm $C_{STRANGLE}$ (in : $k_0, [\mathbf{box}], \epsilon$, inout : $[\mathbf{x}]$)
1	if($[\mathbf{box}].width > \epsilon$)
2	$Bisect([\mathbf{box}], [\mathbf{x_1}], [\mathbf{x_2}])$;
3	$[\mathbf{z}] = CONTRACT(k_0, [\mathbf{x_1}], [\mathbf{x}])$
4	if($[\mathbf{z}]$ has no empty interval)
5	$STRANGLE(k_0, [\mathbf{x_1}], [\mathbf{z}])$;
6	else discard $[\mathbf{z}]$ and paint $[\mathbf{x_1}]$ white;
7	endif
8	$[\mathbf{z}] = CONTRACT(k_0, [\mathbf{x_2}], [\mathbf{x}])$;
9	if($[\mathbf{z}]$ has no empty interval)
10	$STRANGLE(k_0, [\mathbf{x_2}], [\mathbf{z}])$;
11	else discard $[\mathbf{z}]$ and paint $[\mathbf{x_2}]$ white;
12	endif
13	else
14	$[\mathbf{z}] = CONTRACT(k_0, [\mathbf{box}], [\mathbf{x}])$;
15	if($[\mathbf{z}]$ has no empty interval)
16	$[\mathbf{x}] = [\mathbf{x}] \sqcup [\mathbf{z}]$;
17	paint $[\mathbf{box}]$ blue;
18	endif
19	endif

Table 4.2: Strangle algorithm

4.3 Examples

4.3.1 Example 1: Sinusoidal signal

Let's consider an unknown signal $a(t)$ where t is the time. We consider to know only a few properties about a and want to find the smallest tube enclosing the solution. For example, let us consider that a verifies :

$$
\begin{aligned}
a([-\infty;\infty]) &\subset [-1;1] && (4.24)\\
a([\frac{\pi}{2},\pi]) &\subset [-0.7(t-\frac{\pi}{2})+0.99, && (4.25)\\
&\quad -0.1(t-\frac{\pi}{2})+1.01]\\
a(t+\pi) &= -a(t) && (4.26)\\
a(t+2\pi) &= a(t) && (4.27)\\
b(t-\frac{\pi}{2}) &= a(t) && (4.28)\\
b(t) &= \dot{a}(t) && (4.29)
\end{aligned}
$$

We first define the tube $[a](t) = [a^-(t), a^+(t)] = [-\infty, \infty]$ then apply contractors 4.24 and 4.25. Each inclusion actually represents two contractors. For instance, 4.25 is equivalent to

$$
\forall t \in [\frac{\pi}{2},\pi], \quad \begin{array}{l} a(t) \leqslant -0.1(t-\frac{\pi}{2})+1.01 \\ a(t) \geqslant -0.7(t-\frac{\pi}{2})+0.99 \end{array} \tag{4.30}
$$

Therefore, we apply the superiority contractor presented in section 3. The result is presented Fig. 4.5(a). Contractors 4.26 and 4.27 respectively shows that our signal is symmetrical with respect to the point $(\pi, 0)$ and is 2π periodic. We apply the symmetry and periodicity contractors. The results of the contractions are presented Fig. 4.5(b) and Fig. 4.5(c).

Then we apply the time delayed contractor on 4.28 to create a new tube $[b]$:

$$
[b(t)] = [b(t)] \cap [a(t+\frac{\pi}{2})] \tag{4.31}
$$

Finally, from the differential equation 4.29, we can integrate $[b]$ and contract $[a]$ using the equality contractor :

$$
[a(t)] = [a(t)] \cap [\int_{\tau=0}^{t} \frac{db^+(\tau)}{dt} d\tau, \int_{\tau=0}^{t} \frac{db^-(\tau)}{dt} d\tau] \tag{4.32}
$$

The results is illustrated in Fig. 4.5(d). To contract the tube even more, we can re-apply all the contractors one time (Fig. 4.5(e)), three times (Fig. 4.5(f)), five times (Fig. 4.5(g)) or just until the width of the tube is satisfying or until a fixed point is reached (Fig. 4.5(h)). We here clearly recognize the sine signal.

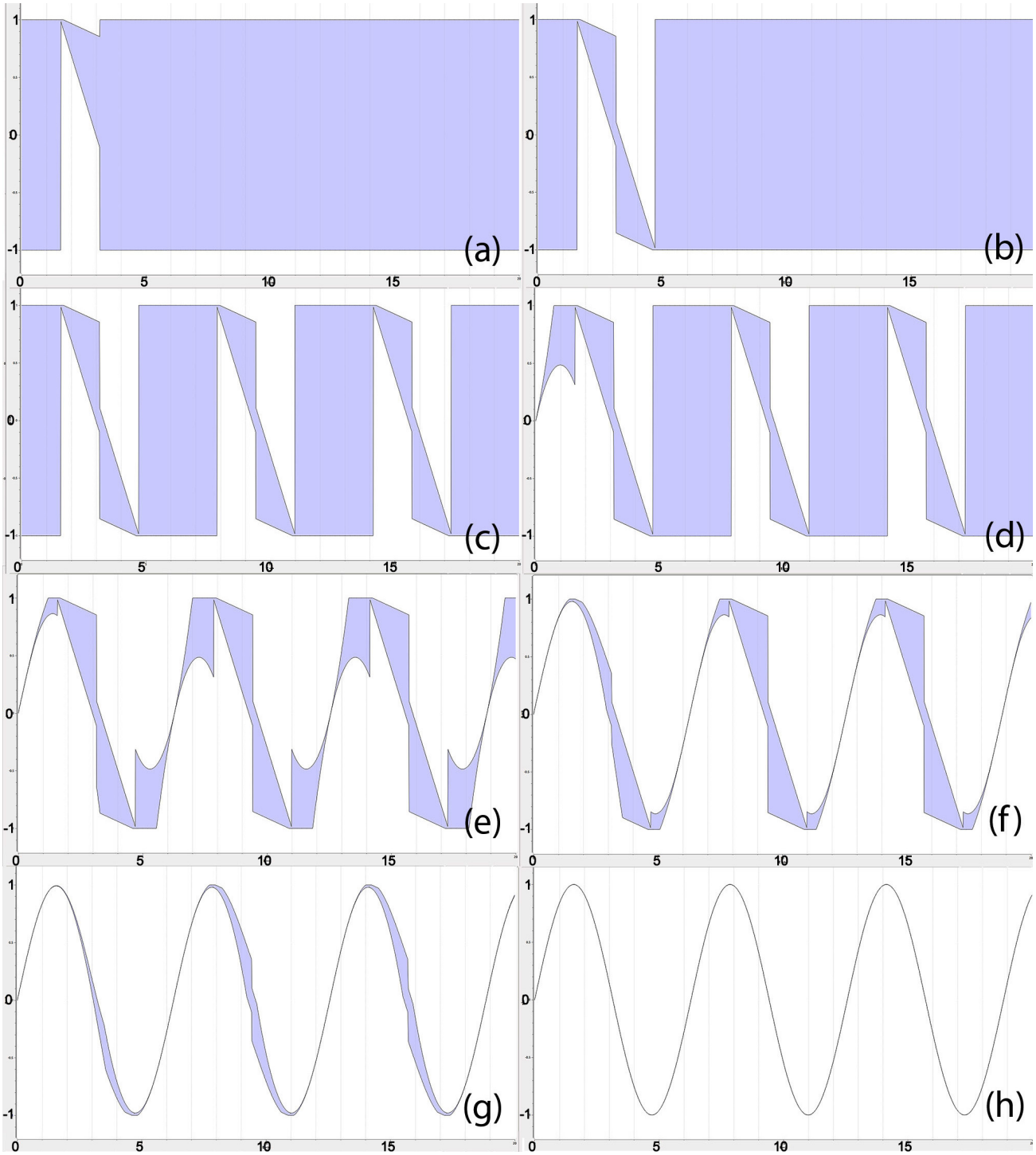

Figure 4.5: Successive contraction of the tube $[a]$.

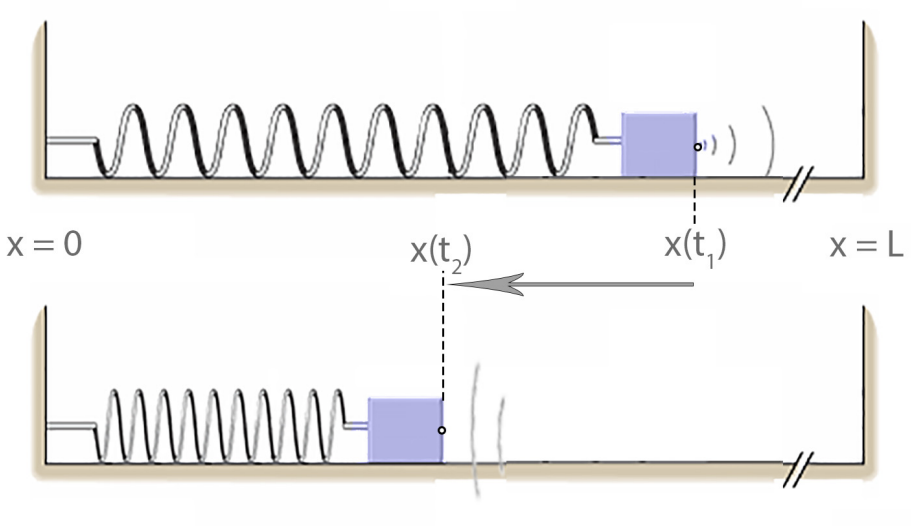

Figure 4.6: Mass-Spring-Sonar system

For the sake of transparency, we used a window size of 20s, a step of 0.01s. Vector size : 20000 intervals represented as two doubles. Contractions are made linearly (sliding interval) from 0s to 20s. The computation time is directly proportional to the size of the computation window. The code was realised in C++ with no optimisation. Over 10 runs of the algorithm, the computation time on a single 3.2Ghz processor is on average 7 ms for Fig. 4.5(d) and 55 ms for Fig. 4.5(h), and respectively use 250Ko and 2Mo of memory.

4.3.2 Example 2: Non-linear mass-spring system

Let us now consider an example based on the famous academic problem that is the mass-spring system presented in Fig.4.6. In actual systems, there is a progressive stiffening or weakening of the spring in x^3 as it is elongated or compressed. This causes the response of the system to be non-linear. Its dynamics is given by *Newton's second law*:

$$m.\ddot{x} + \gamma.\dot{x} + \kappa.x - \beta x^3 = 0 \tag{4.33}$$

where β is the stiffness of the spring, m is the mass, x is the displacement, κ is the spring elasticity and γ is the damping constant.

Knowing the initial conditions, this system could be easily solved using standard numerical methods. However, the particularity of our approach is to consider that we do not know the initial conditions. In return, we equip the mass with a acoustic modem that sends a ping every second. We consider the sound wave sent at t_1 to travel at $c = 100m.s^{-1}$ to the right wall, where it is reflected and received back to the acoustic modem at t_2. In our simulation, we place the right wall at $L = 10m$. This means that the mass moves significantly between the emission and the reception of pings. Each ping is represented by an inter-temporal equation:

$$\begin{array}{rcl} (L - x(t_1)) + (L - x(t_2)) & = & c.(t_2 - t_1) \\ \Leftrightarrow x(t_2) + x(t_1) & = 2L- & c.(t_2 - t_1) \end{array} \tag{4.34}$$

To our knowledge, we cannot easily solve this nonlinear system with standard numerical methods. We thus consider this problem as a state estimation problem of a non-linear delayed system and use our constraint propagation approach defined in the previous section. We first rewrite the system as state equations :

$$\begin{array}{ll} \frac{d}{dt}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \frac{(\beta x^2 - \kappa).x - \gamma.\dot{x}}{m} \end{pmatrix} & \text{(Evolution equation)} \\ x(t_i) + x(t_j) = 2L - c.(t_i - t_j) & \text{(Observation equations)} \end{array} \tag{4.35}$$

where the t_j are sending times and t_i are receiving times. Each ping represents an asynchronous constraint that translates into a contractor for $[x]$ and $[\dot{x}]$.

We developed a simulator using *MATLAB's Ode45* and defined an arbitrary initial state. The result is presented Fig.4.7. We then simulated acoustic pings every seconds. We also simulated errors within known bounds in the measurements up to $+/-0.05m$ to stay consistent with guaranteed results. We also purposefully "lost" a few pings to simulate sporadic measurements. For the sake of transparency, we used $m = 8kg$, $\gamma = 1$, $\kappa = 2$, $\beta = -0.5$ and $L = 100m$. Initial conditions are $x_0 = 10m$ and $\dot{x}_0 = 0m.s^{-1}$.

The simulated observations have the following form:

$$\begin{array}{rcl} x(3.00) + x(3.48) & = & 34.33 \\ x(9.00) + x(9.54) & = & 15.46 \\ x(12.00) + x(12.55) & = & 13.06 \\ x(20.00) + x(20.51) & = & 24.38 \\ & & etc. \end{array} \tag{4.36}$$

We call i the numerical discretized version of t_i and respectively j for t_j. In order to demonstrate the procedure numerically, let's choose a simple *Euler method*, 4.35 then becomes :

$$\begin{array}{rcl} x(k+1) & = & x(k) + dt.\dot{x}(k) \\ \dot{x}(k+1) & = & \dot{x}(k) + dt.\dfrac{(\beta x^2(k) - \kappa).x(k) - \gamma.\dot{x}(k)}{m} \\ x(i) + x(j) - l & = & 0 \end{array} \tag{4.37}$$

where $l = 2L - c.(i - j)$ which is a small measured interval. Therefore we have :

$$\begin{aligned} x(t_j) &= x(j-1) + dt.\dot{x}(j-1) \\ &= x(j-2) + dt.\dot{x}(j-2) + dt.\dot{x}(j-1) \\ &= x(j-3) + dt.\dot{x}(j-3) + dt.\dot{x}(j-2) + dt.\dot{x}(j-1) \\ &= x(i) + dt.\sum_{k=i}^{j-1}\dot{x}(k) \end{aligned} \tag{4.38}$$

and

$$x(j) - x(i) = v \tag{4.39}$$

$$x(j) + x(i) = l \tag{4.40}$$

where $v = dt.\sum_{k=i}^{j-1}\dot{x}(k)$ which is equivalent to

$$\begin{aligned} x(i) &= \frac{1}{2}(l-v) \\ x(j) &= \frac{1}{2}(l+v) \end{aligned} \tag{4.41}$$

To sum up, we have the following contractors :

∘ *State equation.* Tubes $[x]$ and $[\dot{x}]$, contracted using the evolution equation from 4.35:

$$\begin{aligned} [\dot{x}(t)] &= [\dot{x}(t)] \cap \int_{\tau=0}^{t} \left[\frac{(\beta x^2 - k).x - c.\dot{x}}{m}\right](\tau).d\tau \\ [x(t)] &= [x(t)] \cap \int_{\tau=0}^{t} [\dot{x}](\tau).d\tau \end{aligned} \tag{4.42}$$

∘ *Movements.* Using the speed tube $[\dot{x}]$ of the mass, we can compute all the $n_{\max}$ variables $v_n = dt.\sum_{k=i_n}^{j_n-1}\dot{x}(k)$, where n corresponds to the n-th acoustic ping and v_n corresponds to the movement of the mass between the n-th acoustic emission and n-th acoustic reception. We have the following contractor:

$$[v_n] = [v_n] \cap dt.\sum_{k=i_n}^{j_n-1}[\dot{x}](k) \tag{4.43}$$

∘ *Positions.* From 4.41, we get two more contractors:

$$\begin{aligned} [x](i_n) &= [x](i_n) \cap \frac{1}{2}([l_n] - [v_n]) \\ [x](j_n) &= [x](j_n) \cap \frac{1}{2}([l_n] + [v_n]) \end{aligned} \tag{4.44}$$

where i_n is the discretized time of emission t_i of the ping n and j_n is the discretized time of emission t_j.

The algorithm $STRANGLE$ and $CONTRACT$ algorithms are generic versions meant to explain the principle on one dimension. The algorithm $STRANGLE2$ and $CONTRACT2$ presented in table 4.3 and 4.4 are the versions specifically adapted to the mass-spring-sonar example, meant to work simultaneously on two dimensions, the position x and speed $\dot{x}$. Given an initial [**box**] $= [0, 100] \times [-100, 100]$ for $[x]$ and $[\dot{x}]$ at an arbitrary chosen time k_0, $STRANGLE2$ bisects [**box**] and call $CONTRACT2$ on both sub-boxes. $CONTRACT2$ then apply the contractors 4.43 and 4.44, 4.42 and propagates the evolution equation forward then backward along the tubes. If the contracted tubes contain empty intervals, the given sub-box at k_0 does not contain the solution. In this case, we discard the sub-box (white area on Fig.4.9). If the contracted tubes have no empty interval, then the tubes might contain the solution (blue area on Fig.4.9). The sub-box become the new [**box**] and $STRANGLE2$ is called again until a given width ϵ (or precision) is reached. $STRANGLE2$ then takes the union of all non-empty tubes. The union is guaranteed to contains the solution (in respect to the integration method).

Notice that because of the wrapping effect on each step of the propagation, the tubes $[x]$ and $[\dot{x}]$ rapidly explode. It is therefore impossible to contract efficiently the tubes from start ($k_0 = 0$) to end ($k_0 = \bar{k}$) in one passing. Therefore, $STRANGLE2$ have to be called for multiple time k_0 between 0 and $\bar{k}$.

Fig.4.8 shows the result of the algorithm. (1p) represents the position tube $[x]$ after applying the contractors 4.44 and (1s) represents the speed tube $[\dot{x}]$. (2p) and (2s) represent the same tubes after applying $STRANGLE2$ for k_0=0s. (3p) and (3s) for $k_0 = 0s$, $k_0 = 25s$ and $k_0 = 50s$. The final result is shown in (4p) and (4s) for which $STRANGLE2$ is applied for all k_0 such that $k_0\%5 = 0s$.

Just as the $STRANGLE2$ and $CONTRACT2$ are specific versions for the mass-spring-sonar system, the $STRANGLE$ and $CONTRACT$ algorithms can be adapted to constraint satisfaction problems that has a differential evolution equation, and inter-temporal observation functions. For the sake of transparency, we used a window size of 60s and a step of 0.01s. Vector size : 60000 intervals represented as two doubles. For each call to the $STRANGLE$ algorithm, contractions are made linearly (sliding interval) from 0s to 60s, then again from 60s to 0s. The computation time is directly proportional to the size of the computation window. Code realised in C++ with no optimisation. Over 10 runs of the algorithm, the computation time for 4.8(4p) and (4s) on a single 3.2Ghz processor is on average 2.5s and uses 7Mo of memory.

4.3.3 Example 3: Group of AUVs

Let us now consider a group of AUVs. When not submerged, the AUVs are able to use the GPS to accurately compute their position. However, when they are underwater, they can no longer use the GPS and have to estimate their position using their state equations. To illustrate the method, we developed a 3D simulator, namely *SwarmX* for *Swarm explorer*, that generates a set of data from a simulated group

	Algorithm $C_{CONTRACT2}$ (in : $k_0, [\mathbf{box}]$, inout: $[\mathbf{x}], [\dot{\mathbf{x}}]$)
1	for $k := 0$ to $\bar{k}$
2	if($k == k_0$)
3	$[\mathbf{x}](k) = [\mathbf{box}](1)$;
4	$[\dot{\mathbf{x}}](k) = [\mathbf{box}](2)$;
5	endif
6	$[\mathbf{x}](k+1) = [\mathbf{x}](k+1) \cap [\boldsymbol{\phi}_1]([\mathbf{x}](k), [\dot{\mathbf{x}}](k))$;
7	$[\dot{\mathbf{x}}](k+1) = [\dot{\mathbf{x}}](k+1) \cap [\boldsymbol{\phi}_2]([\mathbf{x}](k), [\dot{\mathbf{x}}](k))$;
8	if($k == i_n$ or $k == j_n$)
9	$[v_n] = [v_n] \cap dt. \sum_{p=i_n}^{j_n-1} [\dot{x}](p)$;
10	$[\mathbf{x}](i_n) = [\mathbf{x}](i_n) \cap \frac{1}{2}([l_n] - [v_n])$;
11	$[\mathbf{x}](j_n) = [\mathbf{x}](j_n) \cap \frac{1}{2}([l_n] + [v_n])$;
12	endif
13	endfor
14	for $k := \bar{k}$ to 0
15	if($k == k_0$)
16	$[\mathbf{x}](k) = [\mathbf{box}](1)$;
17	$[\dot{\mathbf{x}}](k) = [\mathbf{box}](2)$;
18	endif
19	$[\mathbf{x}](k) = [\mathbf{x}](k) \cap [\tilde{\boldsymbol{\phi}}_1]([\mathbf{x}](k+1), [\dot{\mathbf{x}}](k+1))$;
20	$[\dot{\mathbf{x}}](k) = [\dot{\mathbf{x}}](k) \cap [\tilde{\boldsymbol{\phi}}_2]([\mathbf{x}](k+1), [\dot{\mathbf{x}}](k+1))$;
21	if($k == i_n$ or $k == j_n$)
22	$[v_n] = [v_n] \cap dt. \sum_{p=i_n}^{j_n-1} [\dot{x}](p)$;
23	$[\mathbf{x}](i_n) = [\mathbf{x}](i_n) \cap \frac{1}{2}([l_n] - [v_n])$;
24	$[\mathbf{x}](j_n) = [\mathbf{x}](j_n) \cap \frac{1}{2}([l_n] + [v_n])$;
25	endif
26	endfor
27	return $([x], [\dot{x}])$;

Table 4.3: Contract algorithm for Ex. 2

	Algorithm $C_{STRANGLE2}$ (in : $\epsilon, k0, [\mathbf{box}]$, inout : $[\mathbf{x}], [\dot{\mathbf{x}}]$)
1	if($box.width > \epsilon$)
2	$Bisect([\mathbf{box}], [\mathbf{subbox1}], [\mathbf{subbox2}])$;
3	$([\mathbf{z}], [\dot{\mathbf{z}}]) = CONTRACT2([\mathbf{subbox1}], [\mathbf{z}], [\dot{\mathbf{z}}])$
4	if($[\mathbf{z}]$ and $[\dot{\mathbf{z}}]$ have no empty interval)
5	$STRANGLE2([\mathbf{subbox1}], [\mathbf{z}], [\dot{\mathbf{z}}])$;
	else discard $([\mathbf{z}], [\dot{\mathbf{z}}])$ and paint $[\mathbf{subbox1}]$ white;
6	endif
7	$([\mathbf{z}], [\dot{\mathbf{z}}]) = CONTRACT2([\mathbf{subbox2}], [\mathbf{z}], [\dot{\mathbf{z}}])$
8	if($[\mathbf{z}]$ and $[\dot{\mathbf{z}}]$ have no empty interval)
9	$STRANGLE2([\mathbf{subbox2}], [\mathbf{z}], [\dot{\mathbf{z}}])$;
	else discard $([\mathbf{z}], [\dot{\mathbf{z}}])$ and paint $[\mathbf{subbox2}]$ white;
10	endif
11	else
12	$([\mathbf{z}], [\dot{\mathbf{z}}]) = CONTRACT2([\mathbf{box}], [\mathbf{z}], [\dot{\mathbf{z}}])$
13	if($[\mathbf{z}]$ and $[\dot{\mathbf{z}}]$ have no empty interval)
14	$[\mathbf{x}] = [\mathbf{x}] \sqcup [\mathbf{z}]$;
15	$[\dot{\mathbf{x}}] = [\dot{\mathbf{x}}] \sqcup [\dot{\mathbf{z}}]$;
	paint $[\mathbf{box}]$ blue;
16	endif
17	endif

Table 4.4: Strangle algorithm for Ex. 2

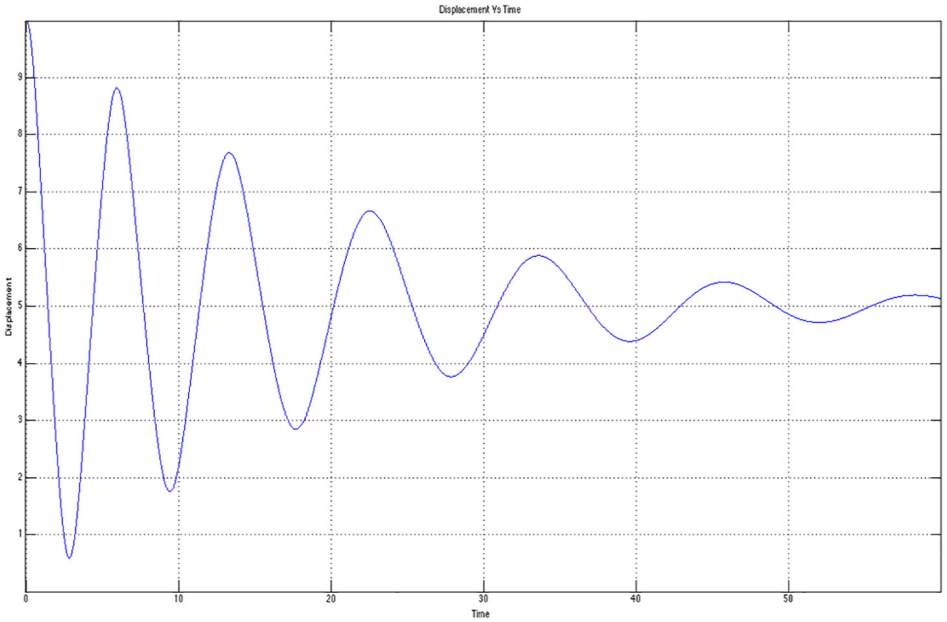

Figure 4.7: MATLAB's Ode45 solution for the mass-spring-sonar system. Notice that Ode45 is only used to simulate the system using the differential equation and the initial conditions. Then a sonar is simulated, and from the solution provided by Ode45, inter-temporal measurements are generated. Without using the initial conditions (supposed unknown), but based solemnly on the differential equation and the inter-temporal measurements, our interval approach then allows us to find a tube around the solution.

of 6 robots following given trajectories. The controller of each robot aims to make it follow the following trajectories :

$$\forall i \in \{1,2,3\}, \hat{\mathbf{x}}_i = \begin{pmatrix} 100\sin t \\ 100\cos t \\ (10\cos t) - 10 \end{pmatrix} \tag{4.45}$$

$$\forall i \in \{4,5,6\}, \hat{\mathbf{x}}_i = \begin{pmatrix} 100\sin t \\ 100\cos t \\ (10\cos t) - 30 \end{pmatrix} \tag{4.46}$$

We assume that the state of each AUV is described by the following equations :

$$\begin{aligned} \dot{x}_1 &= u_1 \cos u_2 \cos u_3 \\ \dot{x}_2 &= u_1 \cos u_2 \sin u_3 \\ \dot{x}_3 &= -u_1 \sin u_2 \end{aligned} \tag{4.47}$$

where the speed u_1, yaw u_2 and pitch u_3 are the entrance of the system. The roll is not commanded but the robot is weighted so that it naturally tends to zero. The initial state is unknown. The estimated position is represented by a box enclosing the real position of each AUV. This box is obviously very thin when the

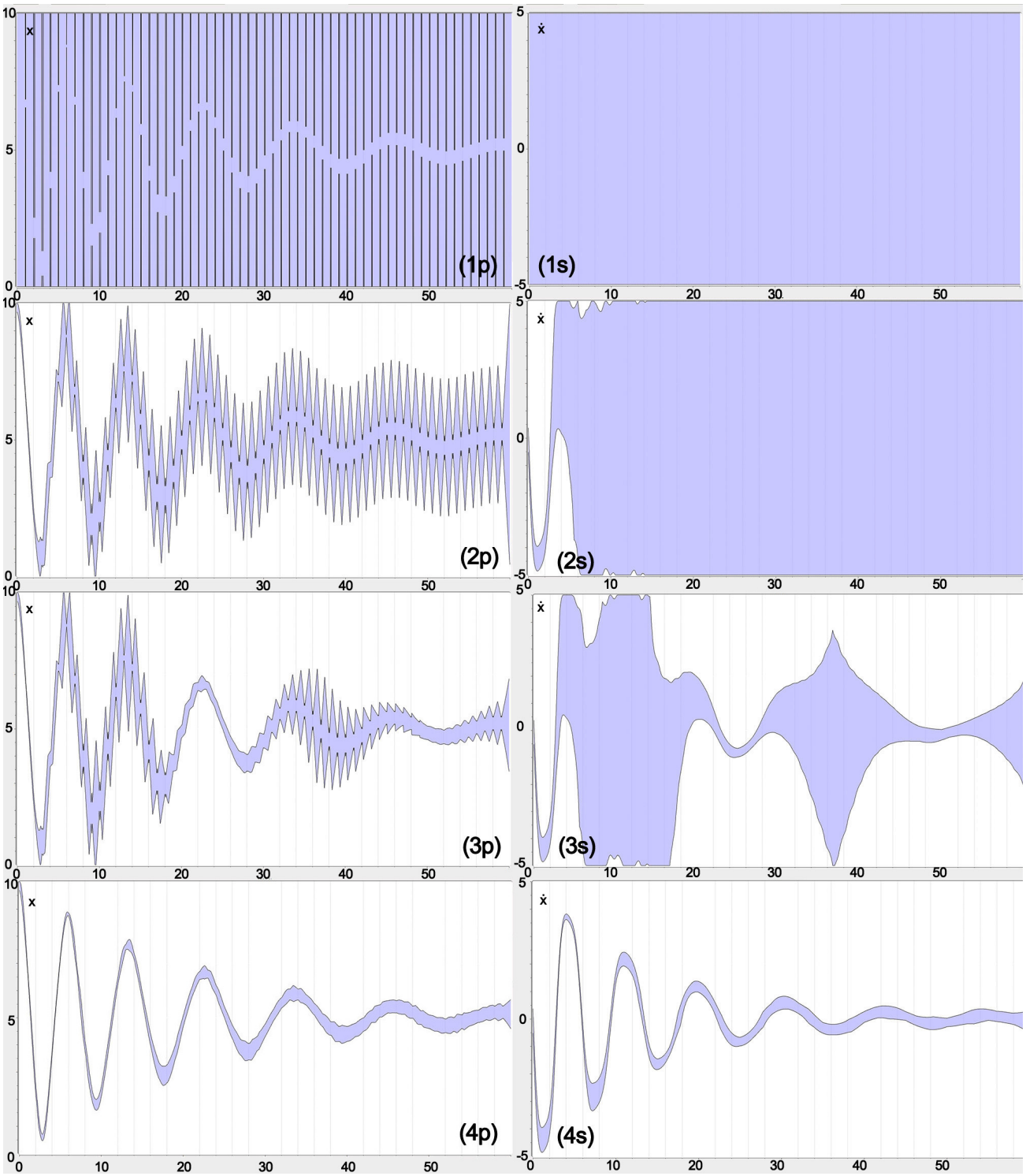

Figure 4.8: Successive applications of the $STRANGLE2$ algorithm: (1) before application, (2) at $k_0 = 0s$, (3) at $k_0 = 0s$, $k_0 = 25s$ and at $k_0 = 50s$, and (4) for all k_0 such that $k_0 \% 5 = 0s$. (p) is the position and (s) is the speed.

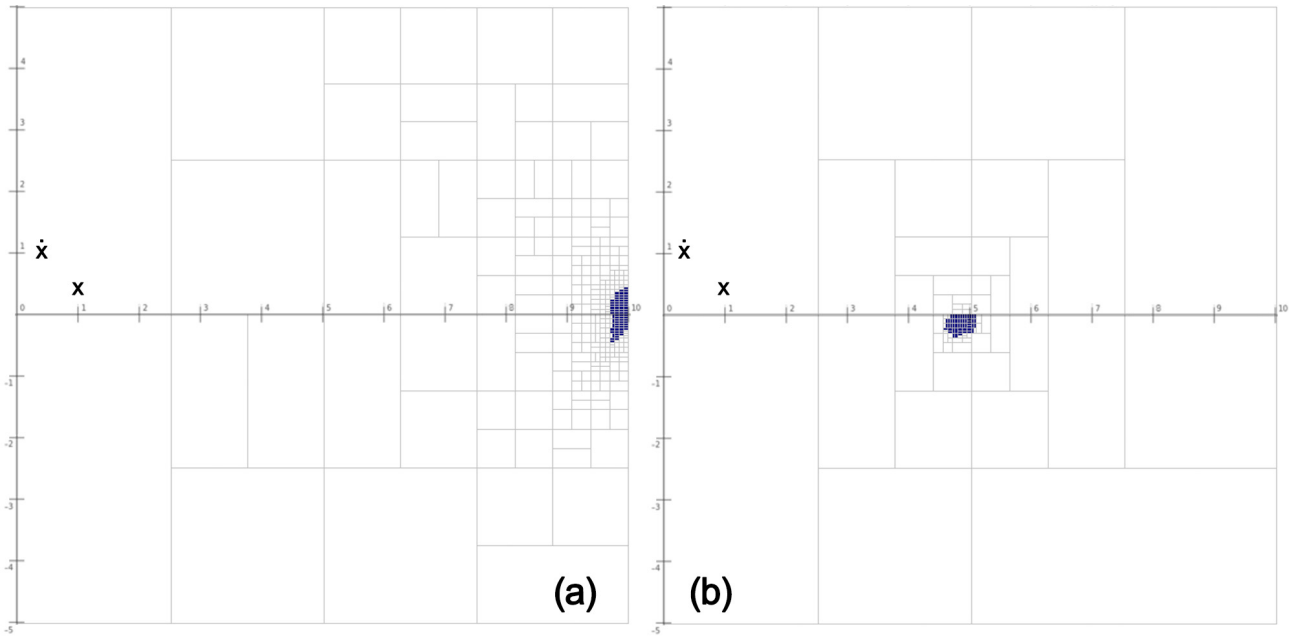

Figure 4.9: Result on an $x, \dot{x}$ plane of the $STRANGLE2$ bisections at (a) $t = 0s$ and (b) $t = 50s$. The white boxes represent tubes that get empty at some point. The blue boxes represent the tubes that never have empty interval for all t. The solution is in the union of all these tubes. These figures can be interpreted as a "slice" of the $([\mathbf{x}], [\dot{\mathbf{x}}])$ tubes at a given time. Notice that the slice at $t = 0s$ gives us the initial conditions.

AUV uses the GPS, then gets bigger and bigger underwater. AUVs 1,2 and 3 have a trajectory that reaches the surface at some point, so they can contract their position box using the GPS. AUVs 4,5 and 6 never reach the surface but their trajectory place them in range of acoustic communication with AUVs 1,2 and 3. The idea is to contract these boxes when two or more AUVs are in range of communication with each other. We equipped each submarine with a acoustic modem that can broadcast the estimated position box of the AUV every second. The particularity of our approach is to consider that the distance measurements are not instantaneous. The sonar waves move at the celerity c of the sound underwater. As AUVs are often slow compared to c and close to each other, we can often neglect the time for a sonar wave to go from an AUV to another. Here we won't make this approximation. Therefore we cannot suppose that we measure a true distance between AUVs at the same time, but between AUVs at different times. Note that the communication is one-way only and does not have to be synchronized between the AUVs. When AUV i received at t_i a acoustic ping from robot j emitted at t_j, we get a constraint of the form :

$$||\mathbf{p}_i(t_i) - \mathbf{p}_j(t_j)|| = c.(t_i - t_j) \tag{4.48}$$

where c is the celery of the sound in water and $\mathbf{p}_i$ is the position vector of a submarine i in x-y-z coordinates. Therefore, we have a system of constraints similar to the example 2. Fig.4.10 presents the 3D simulator developed to generate the constraints for each AUV, e.g.:

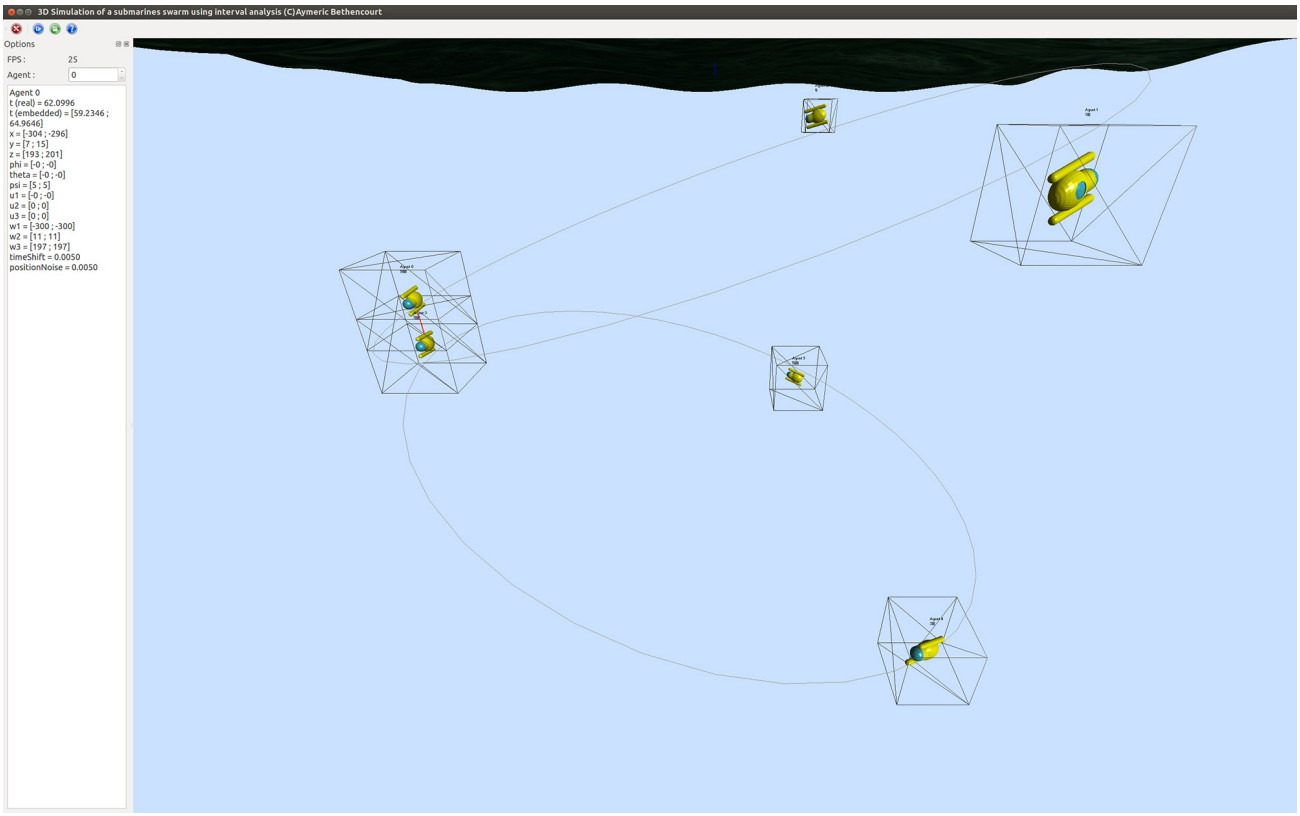

Figure 4.10: SwarmX v1 : 3D simulation of a group of 6 AUVs. AUVs 1,2 and 3 can contract their position box when reaching the surface, and AUVs 4,5 and 6 can contract their position box when communicating with other AUVs (communication symbolized by a red line between AUVs). A video of the simulation is available on http://youtu.be/0cjzzsaWTvA.

$$||p_1(1.00) + p_2(1.32)|| = 128.44 \quad (4.49)$$
$$||p_1(9.00) + p_3(9.74)|| = 284.38 \quad (4.50)$$
$$||p_3(9.00) + p_5(9.15)|| = 72.49 \quad (4.51)$$
$$||p_5(17.00) + p_4(18.13)|| = 682.98 \quad (4.52)$$
$$etc.$$

Using the same approach than in the example 2, we can solve this problem using constraint propagation on tubes. Fig. 4.11 shows the result of the contractions for the position of an AUV.

For the sake of transparency, we used a window size of 100s and a step of 0.1s. Vector size : 10000 intervals represented as two doubles. The contractions are made linearly (sliding interval) from 0s to 100s for 4.11(b) and again from 100s to 0s for 4.11(c). The computation time is directly proportional to the size of the computation window. Code realised in C++ with no optimisation. Over 10 runs of the algorithm, the computation time on a single 3.2Ghz processor is on average 15ms and uses 900Ko of memory.

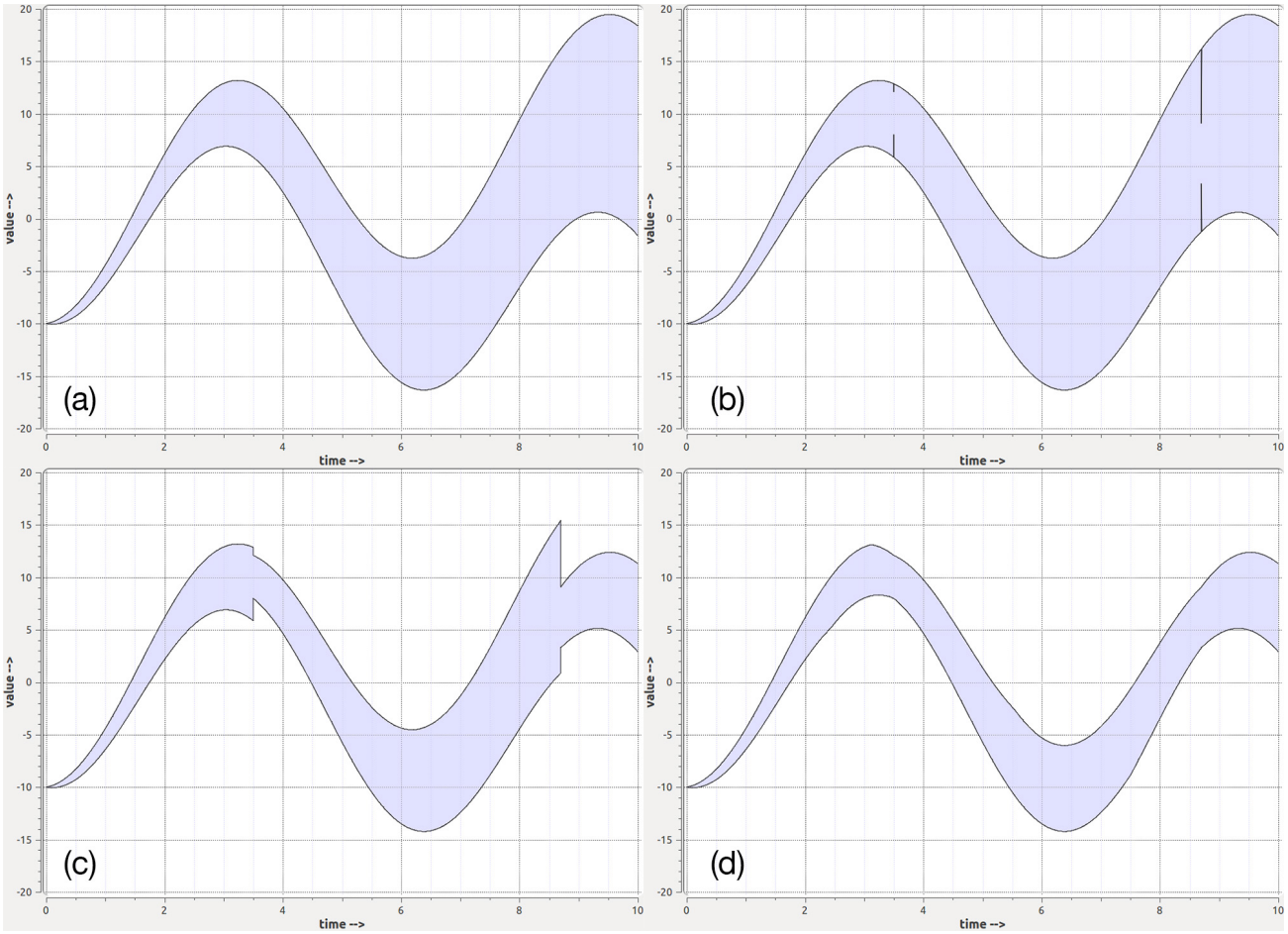

Figure 4.11: Result of the contractions for the abscissa of robot $i = 4$, (a) if no ping is received, (b) if 2 pings is received and we use the contractor associated with the observation, (b) then we use the contractor associated with the evolution with t increasing (real time) (c) then with t decreasing (post-treatment).

4.4 Conclusion

Solving nonlinear systems involving differential equations is a difficult problem, especially when the initial conditions are unknown or when the problem is ill-conditioned, e.g. inter-temporal measurements are involved. To numerically solve this class of problems, this chapter has first introduced the notion of tube which encompasses the informations needed to guarantee associations upon trajectories. Then, an arithmetic was developed around this notion, and a contractor-based approach has followed. As a result, a method has been proposed to contract tubes that enclose the solution. As most interval-based methods, this approach can be combined with probabilistic methods [Abdallah et al., 2008] and made robust with respect to outliers by relaxing a given number of constraints (Chapter 1).

Finally, in order to share our research with the community, we integrated tubes, their properties, operators and minimal contractors presented below within the *IBEX library* and should soon be integrated by default in *IBEX*. In the meantime you can download all source codes for this chapter on `http://aymericbethencourt.com/thesis`.

In the next chapter, we will consider a more realistic approach for the localization problem by considering that the clock of the robots is not properly synchronized and therefore that the time measurements are uncertain.

The work presented in this chapter has been **published** in : *Mathematics in Computer Science, Special Issue on Interval Methods and Applications*, 2014.

Chapter 5

Cooperative localization of underwater robots with unsynchronized clocks

5.1 Introduction

This chapter proposes a new approach to localize a group of AUVs. Localizing AUVs can be particularly difficult underwater as there is no installed infrastructure. As we saw in the previous chapter, AUVs can only access the GPS on the surface as the electromagnetic waves can hardly penetrate water. The only tools available underwater are dead-reckoning and acoustical localization systems. To solve this problem, many probabilistic approaches have been studied [Thrun et al., 2005][Bahr et al., 2006], that usually consider the robots close to each other and moving slowly enough so that the displacement of the acoustic signal can be consider negligible compared to the precision of the localization [Zhang et al., 2009]. The distance between the robots is measured, and the triangular inequalities are solved to localize the robots [Leonard and Durrant-Whyte, 1992]. This, of course, implies that the clocks of each robot is perfectly synchronized [Lin et al., 2005].

In our approach, we use acoustic communication and consider that the travel time of the acoustic waves cannot be neglected, e.g. when the robots are fast and far-spaced. Therefore we cannot suppose that we measure a true distance between robots at the same time, but between robots at different times. Moreover, as the clocks of the robots are unsynchronized, the emitting and receiving times of the acoustic waves are uncertain. We also consider our system to be completely *decentralized*, meaning that each robot will only localize itself using the data received from the few other robots that are in range of communication.

Two approaches can be considered to solve this problem. As previously mentioned, the first approach is probabilistic. If the system is linear, the problem can be solved using for instance a *Kalman Filter*, and if the problem is non-linear, an *Extended Kalman Filter* (EKF) [Ljung, 1979] or a *Sequential Monte Carlo* (SMC) method [Thrun et al., 2005] can be considered. This approach has been proven to be particularly effective for robot localization [Bonnifait and Garcia, 1996][Lacroix et al., 2002].

The second approach is based on set-membership. When the problem is linear, Ellipsoids and Polytopes [Gollamudi et al., 1996] can be particularly efficient. However, for non-linear systems, only intervals

and sub-pavings have been proven to be efficient [Jaulin et al., 2001a]. The advantage of this approach compared to probabilist methods is that the result is guaranteed. A closed non-empty interval is guaranteed to contain the solution. Moreover interval analysis is particularly efficient when the number of equations is far superior to the number of variables, which is often the case in localization applications [Jaulin, 2011][Le Bars et al., 2010]. Notice that [Gning and Bonnifait, 2005] proposes a comparison between *Kalman* filtering and constraints propagation techniques as presented in this thesis.

In brief, this chapter proposes an interval based approach for localizing a group of underwater robots with unsynchronized clocks and therefore deal with the inter-temporal constraints on uncertain times. To solve this problem, we cast the state equations of our robots as a constraint satisfaction problem and use notion of tube presented in the previous chapter to encompass the robots trajectory and clock with their uncertainty. We then use interval propagation to successively contract these intervals around the true position and clock of the robots. A simulation and a real experimental test case are presented.

5.2 Problem Statement

Let us first describe our cooperative localization problem. For a robot i in our group or swarm, consider the state equations of the robot as follows:

$$\begin{array}{cc} \dot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) + \mathbf{n}_\mathbf{x} & \text{(evolution equation)} \\ \mathbf{y}_i = \mathbf{g}(\mathbf{x}_i) & \text{(observation equation)} \end{array} \tag{5.1}$$

where $\mathbf{x}_i$ is the state vector of the robot i, $\mathbf{u}_i$ its inputs or commands, $\mathbf{y}_i$ its outputs or measures, and $\mathbf{f}$ and $\mathbf{g}$ respectively the evolution and observation functions. $\mathbf{n}_\mathbf{x}$ is the state noise. The uncertainty on y_i will be represented by an interval around its unknown value. Notice that the observation function usually depends on the state of the robot at the current time. The originality of our approach is to consider a acoustic communication between the robots, which will be represented as an inter-temporal constraint between the states of the robot at different times. We consider that each ping encloses the estimated position box of the emitting robot, along with the interval around the emitting time in the robot's clock.

Formulation. An *inter-temporal relation* (or *ping* for short) corresponds to a 4-tuple $\mathbf{p} = (a, b, i, j)$ where $a \in \mathbb{R}$ corresponds to the emission time, $b \in \mathbb{R}$ to the reception time, $i \in \{1, \ldots, m\}$ the emitting robot, and $j \in \{1, \ldots, m\}$ the receiver. Due to the causality, we have $b > a$, or equivalently (a, b) is an element of the t-plane [Aubry et al., 2013]. Denote by $\mathbf{p}(k)$ the k^{th} ping, and by t the true time. We denote by $\tau = h_i(t)$ the clock function [Srikanth and Toueg, 1987] which for an absolute time t matches the inner time τ of the robot i. Notice that h_i is strictly increasing when no re-synchronization happens and piecewise increasing when it does. For $i \in \{1, \ldots, m\}$, $t \in \mathbb{R}$ and $k \in \{1, \ldots, k_{\max}\}$, we have:

$$\begin{array}{ll} \text{(i)} & \dot{\mathbf{x}}_i(t) = \mathbf{f}(\mathbf{x}_i(t), \mathbf{u}_i(t)) + \mathbf{n}_\mathbf{x}(t) \\ \text{(ii)} & g\left(\mathbf{x}_{i(k)}(a(k)), \mathbf{x}_{j(k)}(b(k)), a(k), b(k)\right) = 0 \\ \text{(iii)} & \tilde{a}(k) = h_i(a(k)) \\ \text{(iv)} & \tilde{b}(k) = h_j(b(k)) \\ \text{(v)} & \dot{h}_i(t) = 1 + n_h(t) \end{array} \tag{5.2}$$

(i) corresponds to the state equations of the i-th robots. The state noise $\mathbf{n_x}(t)$ is assumed to be bounded.

(ii) is the inter-temporal observation function so that $g : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}$ is here:

$$g(\mathbf{x}_1, \mathbf{x}_2, a, b) = \|\mathbf{x}_1 - \mathbf{x}_2\| - c * (b - a) \tag{5.3}$$

where c is the speed of sound. Using the *Euclidean* norm, (ii) can be re-written as

$$c * (b(k) - a(k)) = \sqrt{\begin{array}{c}(x_{i(k)}(a(k)) - x_{j(k)}(b(k)))^2 \\ +(y_{i(k)}(a(k)) - y_{j(k)}(b(k)))^2\end{array}} \tag{5.4}$$

(iii) $\tilde{a}(k)$ corresponds to the inner time of the robot $i(k)$ (i.e. the emitter has emitted the k^{th} ping). The clock function h_i matches the inner time $\tilde{a}(k)$ to the real time $a(k)$.

(iv) $\tilde{b}(k)$ corresponds to the inner time of the robot $j(k)$ (i.e. the receiver has received the k^{th} ping). The clock function h_j matches the inner time $\tilde{b}(k)$ to the real time $b(k)$.

(v) $n_h(t)$ is the clock noise and is assumed to be bounded.

Notice that for all k, we know exactly: $\tilde{a}(k), \tilde{b}(k), i(k), j(k)$.

The advantage of such formalism is that it can encompass multiple previous formalisms of set-membership localization [Jaulin, 2011][Le Bars et al., 2010] and in addition takes into consideration inter-temporal constraints on uncertain times. To our knowledge, such formalism has never been considered before. This formalism can also be applied to SLAM by considering only one robot and that landmarks are stationary robots. Measurements with the GPS above surface can also be considered as a measurement of the robot to itself. Therefore this formalism is particularly powerful.

When an AUV send a acoustic ping k at $t = a(k)$ received by another AUV at time $t = b(k)$, the distance between the two AUV can be measured as $c * (b(k) - a(k))$ where c is the celerity of the sound. Thus the measurement between the two robots is inter-temporal and between the position of AUV 1 at $a(k)$ and the position of AUV 2 at $b(k)$. Figure 5.1 illustrates the principle. Notice that these times are uncertain, as the clocks of the robots might not be synchronized. We initially only know a tube (which may be very large) around the clock function h of each robot and are going to contract it to re-synchronize the clock. To our knowledge, this problem cannot be easily solved using standard *Monte Carlo* methods.

5.3 Cooperative localization as a constraint satisfaction problem

The cooperative localization problem described in 5.2 can be considered as a constraint satisfaction problem on tubes and thus we shall apply the contractors presented in the previous chapter. Let's consider a group of multiple AUVs. For any AUV j in our group, we can contract the tubes $[\mathbf{x}_j]$ and $[h_j]$ using the integral

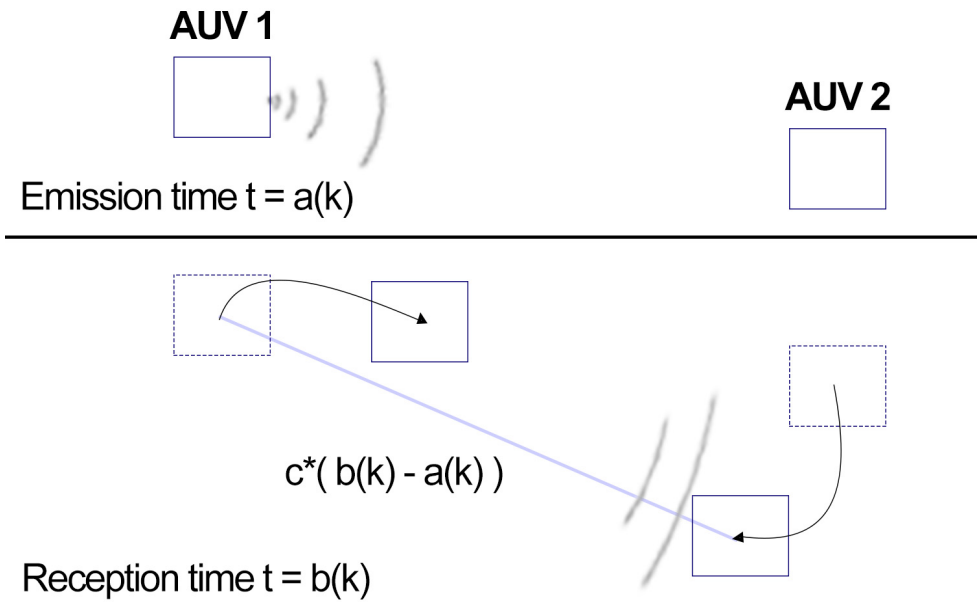

Figure 5.1: When robot 1 sends a sonar signal, it is received by robot 2 after $b(k) - a(k)$ seconds. Both robots have moved during this time. Therefore the measurement is the distance from robot 1 at $a(k)$ to robot 2 at $b(k)$.

contractor from constraints (5.2i) and (5.2v):

$$[\mathbf{x}_j](t) = [\mathbf{x}_j](t) \cap \int_0^t ([\mathbf{f}]([\mathbf{u}_j(\tau)],[\mathbf{x}_j(\tau)]) + [\mathbf{n}_x(\tau)])d\tau \tag{5.5}$$

$$[h_j](t) = [h_j](t) \cap \int_0^t (1 + [n_h](\tau))d\tau \tag{5.6}$$

These contractors are available for any t; and because of the state noise and clock noise, the width of tubes $[\mathbf{x}_j]$ and $[h_j]$ will keep on growing if nothing is done. However when the robot receives a acoustic communication (or ping), equations (5.2ii), (5.2iii) and (5.2iv) will be available as constraints on $[\mathbf{x}_j]$ and $[h_j]$ and thus enable us to contract them punctually at each ping reception.

Let's consider a robot i emitting a ping received by robot j. Each ping contains the estimated position box $[\mathbf{x}_i]$ of the emitting robot, along with the interval around the time $[a(k)]$ robot i sent the ping k in its own clock; so each time j receives a acoustic ping from a robot i, the receiving robot j has data on $[a(k)]$, $[b(k)]$, $[x_{i(k)}](a(k))$, $[x_{j(k)}](b(k))$, $[y_{i(k)}](a(k))$, $[y_{j(k)}](b(k))$ and it can apply a simple forward-backward contractor as presented in Chapter 2 to contract the intervals of :

$$c * ([b(k)] - [a(k)]) = \sqrt{\begin{array}{c}([x_{i(k)}]([a(k)]) - [x_{j(k)}]([b(k)]))^2 \\ +([y_{i(k)}]([a(k)]) - [y_{j(k)}]([b(k)]))^2\end{array}} \tag{5.7}$$

Notice that we chose an arbitrary value for the celerity of the sound c. However it is quite possible to set c as an interval $[c]$ and use the forward-backward algorithm to contract this interval arround its true value.

Fig. 5.2 illustrates the contractions made by the forward-backward algorithm. The purple area represents the interval enclosing the solution $\mathbf{x}_{j(k)}\left(b\left(k\right)\right)$. Before receiving the ping, an interval $[\mathbf{x}_{j(k)}]\left([b\left(k\right)]\right)$ $(before)$ is known from the contracted tube $[\mathbf{x}_i]$ (contractor 5.5). This interval can be contracted to $[\mathbf{x}_{j(k)}]\left([b\left(k\right)]\right)$ $(after)$ by intersecting it with the solutions of (5.2ii), which contracts the position of the robot. Notice that the purple area can also be contracted to the red doted form. This translates in the forward-backward algorithm as a contraction of $[c*(b(k)-a\left(k\right))]$ and thus $[b(k)]$ and the clock as illustrated in the next section. Fig. 5.3 illustrates different cases of contraction.

5.4 Test cases

In all presented test cases, we consider fast and far-spaced AUVs defined by the following evolution equation:

$$\dot{\mathbf{x}}_i(t) = \begin{pmatrix} u_{i1}(t).\cos(u_{i2}(t)) \\ u_{i1}(t).\sin(u_{i2}(t)) \end{pmatrix} + \mathbf{n}_{\mathbf{x}}\left(t\right) \tag{5.8}$$

where $u_{i1}(t)$ and $u_{i2}(t)$ are the components of vector $\mathbf{u}_i(t)$. They are respectively the given speed and steering command to robot i. The vector $\mathbf{x}_i$ contains the abscissa x_i and ordinate y_i of the robot i. $\mathbf{n}_{\mathbf{x}}\left(t\right)$ is the state noise.

5.4.1 Simple example with 2 AUVs.

Let us first demonstrate the contractions with a simple example involving only two AUVs. The AUVs follow circular trajectories such that :

$$\mathbf{x}_1(t) = 10\begin{pmatrix} \cos t \\ \sin t \end{pmatrix} \tag{5.9}$$

$$\mathbf{x}_2(t) = 10\begin{pmatrix} \cos(t+\pi) \\ \sin(t+\pi) \end{pmatrix} \tag{5.10}$$

The state and clock noises are considered higher for the receiving robot than for the emitting robot to clearly illustrate the contraction.

$$\text{For AUV 1, } \forall t, \mathbf{n}_{\mathbf{x}}(t) \in [-0.1, 0.1] \text{ and } n_h\left(t\right) \in [-0.01, 0.01] \tag{5.11}$$

$$\text{For AUV 2, } \forall t, \mathbf{n}_{\mathbf{x}}\left(t\right) \in [-1, 1] \text{ and } n_h\left(t\right) \in [-0.1, 0.1] \tag{5.12}$$

AUV 1 starts at $(10, 0)^T$ and AUV 2 at $(-10, 0)^T$. Fig. 5.4 illustrates this example. We consider the celerity of the sound $c = 100m/s$ to simplify.

Using (5.5) and (5.6) the AUVs can estimate their position and clock over time thanks to the tubes enclosing their real value. However, the width of the tubes will increase indefinitely if nothing is done.

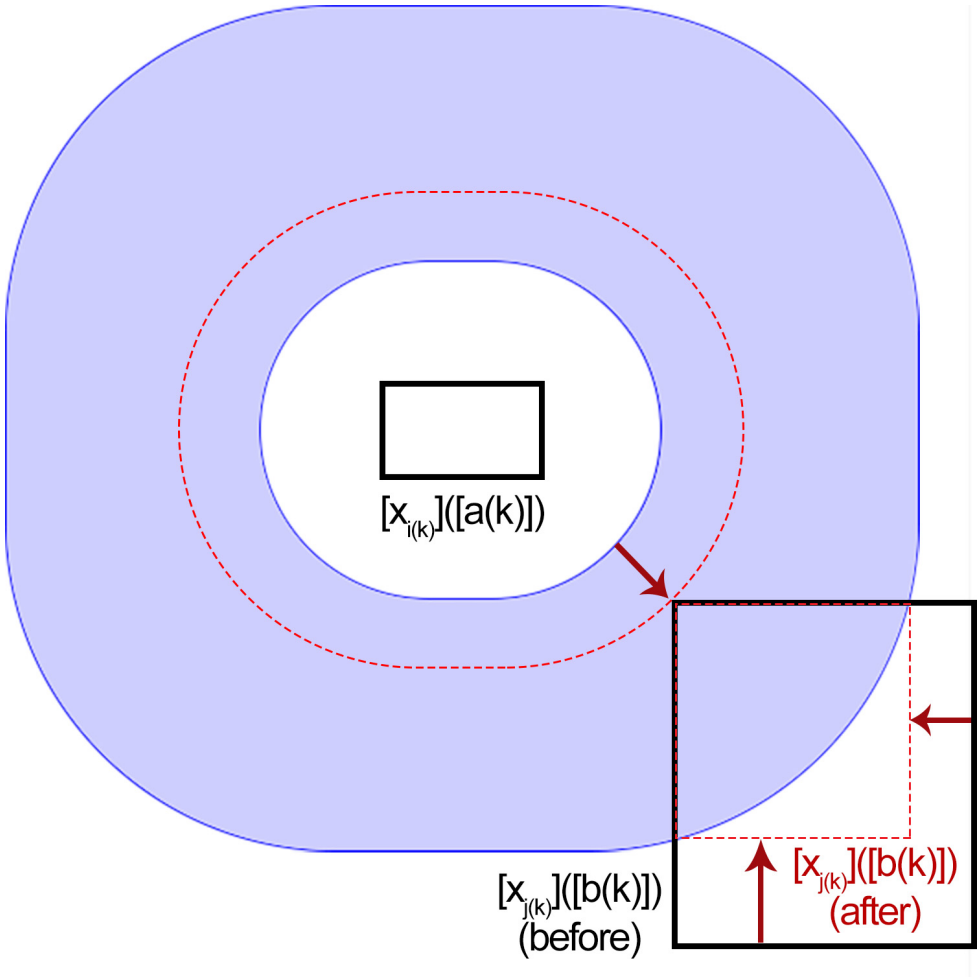

Figure 5.2: Illustration of the contraction of the position box and clock interval. The purple area represents all the possible positions for robot j compatible with the distance measurement to robot i. As robot j already knows a box arround its position before receiving the signal, it can contract his box using the forward-backward contractor. Notice that if there is "some space left" in the purple area (possible contraction to the red doted rounded rectangle), this usually translates in the forward-backward contractor as a contraction on the robot's clock.

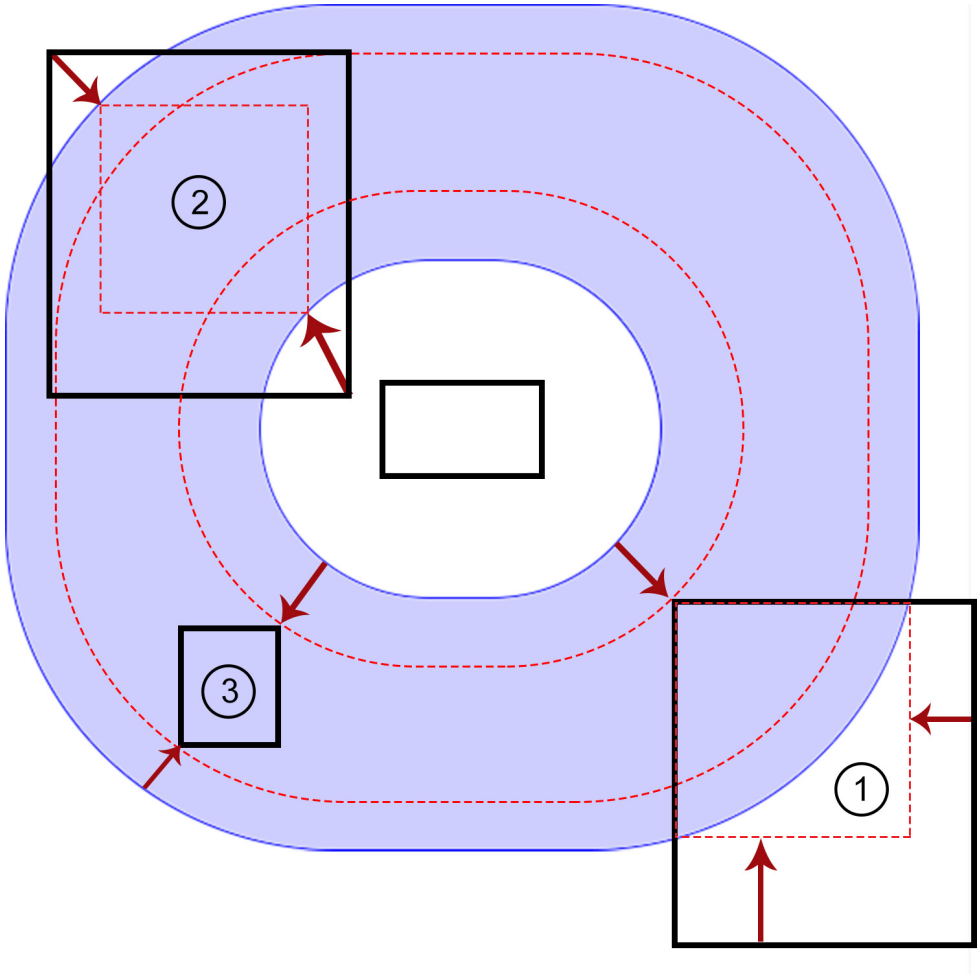

Figure 5.3: (1) Both the position and the clock are contracted. (2) Only the position is contracted. (3) Only the clock is contracted.

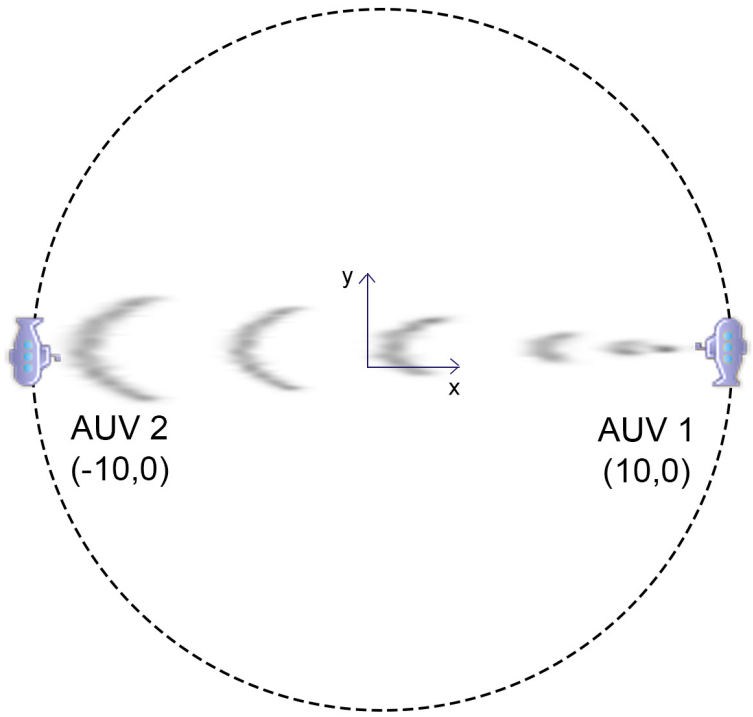

Figure 5.4: Simple example with two AUV moving on a circular trajectory. AUV 1 emits, AUV 2 receives.

AUV 1 sends a acoustic ping $k = 0$ at what it "thinks" is $t = 1s$ in its own clock. This is what we note $\tilde{a}(0) = 1s$ and is not the real time $a(0)$. However using the inverse tube of $[h_1]$, we can get an interval $[a(0)]$ enclosing the real emitting time. Using the tube $[x_1]$ and $[y_1]$, we can then get intervals $[x_1]([a(0)])$ and $[y_1]([a(0)])$ enclosing the position of AUV 1 when emitting the signal. In this example, we obtain:

$\tilde{a}(0)$	$[a(0)]$	$[x_1]([a(0)])$	$[y_1]([a(0)])$
1.00	$[0.99, 1.01]$	$[5.21, 5.59]$	$[8.26, 8.57]$

AUV 2 receives the transmission from AUV 1 at what it "thinks" is $t = 1.37s$. This is what we note $\tilde{b}(0) = 1.37s$ and is not the real time $b(0)$. However using the inverse tube of $[h_2]$, we can get an interval $[b(0)]$ enclosing the real emitting time. Using the tube $[x_2]$ and $[y_2]$, we can then get intervals $[x_2]([b(0)])$ and $[y_2]([b(0)])$ enclosing the position of AUV 2 when receiving the signal. In this example, we obtain:

$\tilde{b}(0)$	$[b(0)]$	$[x_2]([b(0)])$	$[y_2]([b(0)])$
1.37	$[1.22, 1.51]$	$[-4.67, 0.90]$	$[-11.49, -8.16]$

We can then apply our forward-backward algorithm on (5.4) to contract the intervals of AUV 2. The output is as follow (the contracted values have been emphasized):

$\tilde{b}(0)$	$[b(0)]$	$[x_2]([b(0)])$	$[y_2]([b(0)])$
1.23	$[1.22, \mathbf{1.23}]$	$[-4.67, \mathbf{-1.25}]$	$[-11.49, \mathbf{-9.98}]$

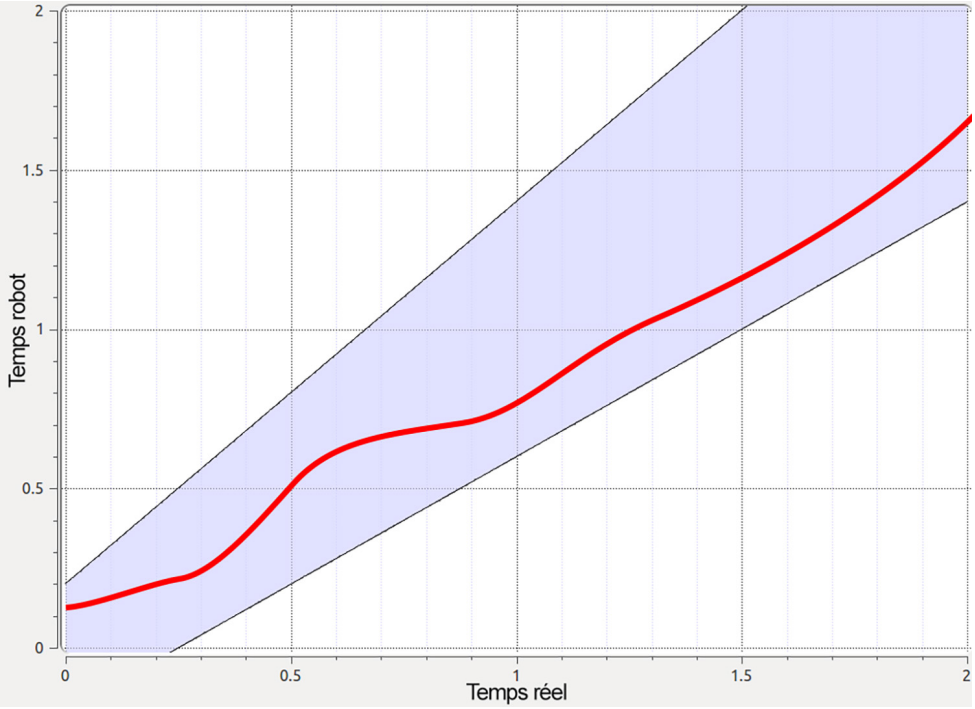

Figure 5.5: Illustration of an hypothetical clock function in red and the tube that encloses it.

Notice that the box around the position of AUV 2 at the reception of the ping has been contracted. The interval enclosing the real time of reception was also contracted which allows us to re-synschronize the clock from $1.37s$ to $1.23s$ at the reception of the signal. We then use the integral contractors (5.5) and (5.6) to propagate the constraint to the rest of the tube. The result of this simulation and the step-by-step procedure is explained in fig. 5.6 and fig. 5.7.

5.4.2 Full simulation with 6 AUVs.

Let us now consider a group a 6 simulated AUVs following two circular trajectories:

$$\forall i \in \{1,2,3\}, \mathbf{x}_i(t) = 10\begin{pmatrix}\sin t - 10\\ \cos t\end{pmatrix} \tag{5.13}$$

$$\forall i \in \{4,5,6\}, \mathbf{x}_i(t) = 10\begin{pmatrix}\sin t\\ \cos t\end{pmatrix} \tag{5.14}$$

All other parameters are the same than in Example 1. Each robot is equipped with a simulated data pinger with a range of $5m$ emitting its position box and clock interval every 10 seconds. Any robot at a distance more than this range will not receive the ping. We suppose that AUVs $i = 1, 2$ and 3 go to the surface when $x_i(t) < -9m$. Therefore they can access the GPS to contract their position box and clock interval. AUV $4, 5$ and 6 however never go to the surface and have to localize themselves using only the received pings from other AUVs. The simulation is presented in Fig. 5.8 and results for AUV 4 in Fig. 5.9.

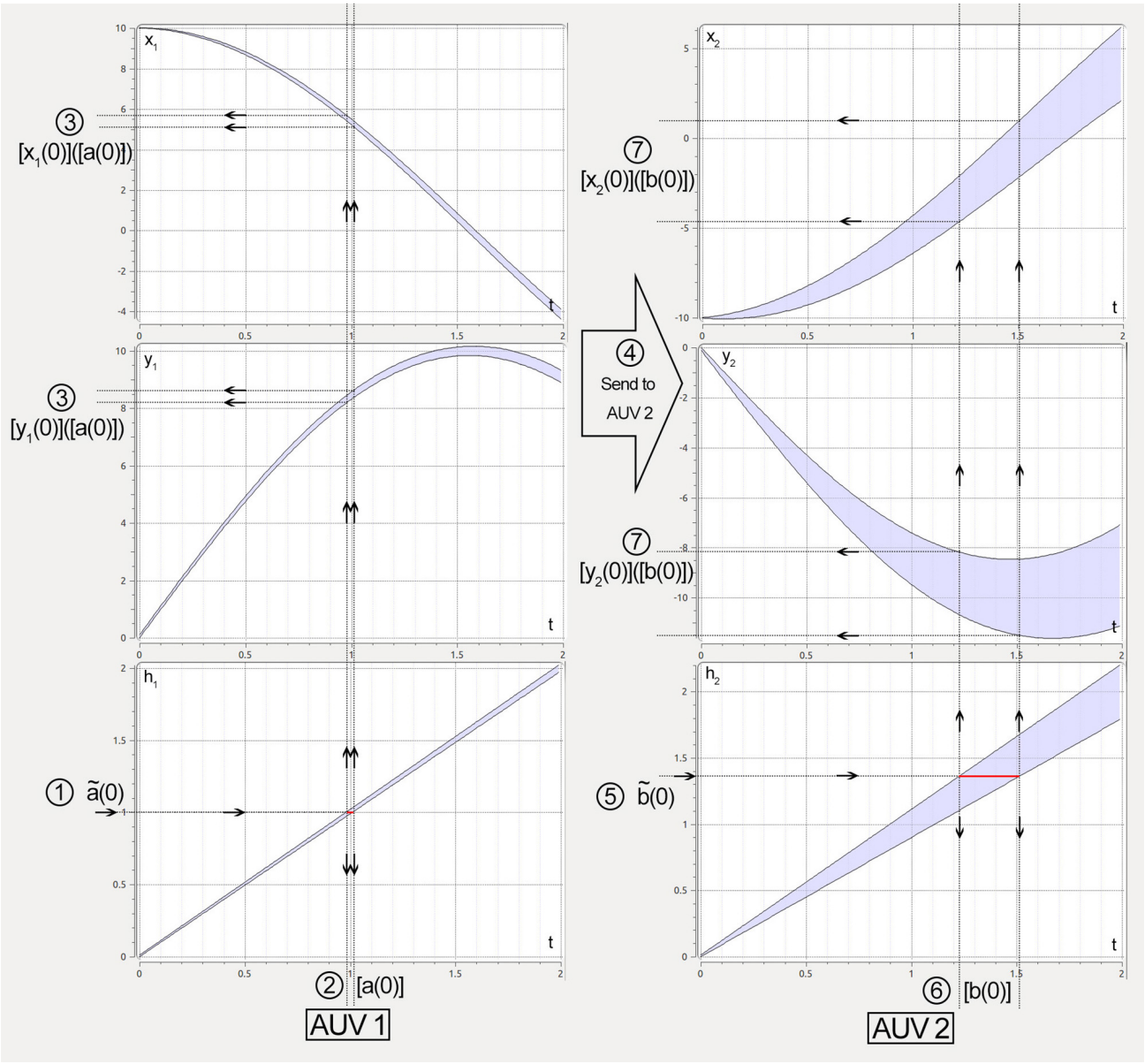

Figure 5.6: (1) AUV 1 sends a ping at what it thinks is $\tilde{a}(0)$ in its own clock. (2) We use the tube inverse of $[h_1]$ to compute that the ping was actually sent at a time enclosed in $[a(0)]$. (3) We use the tubes $[x_1]$ and $[y_1]$ to compute the position of AUV 1 when sending the ping. (4) The ping sent to AUV 2 contains the data $[a(0)]$, $[y1(0)]([a(0)])$ and $[x1(0)]([a(0)])$. (5) AUV 2 receive the ping at what he thinks is $\tilde{b}(0)$ in its clock. (6) We use the tube inverse of $[h_2]$ to compute that the ping was actually received at a time enclosed in $[b(0)]$. (7) We use the tubes $[x_2]$ and $[y_2]$ to compute the position of AUV 2 when receiving the ping.

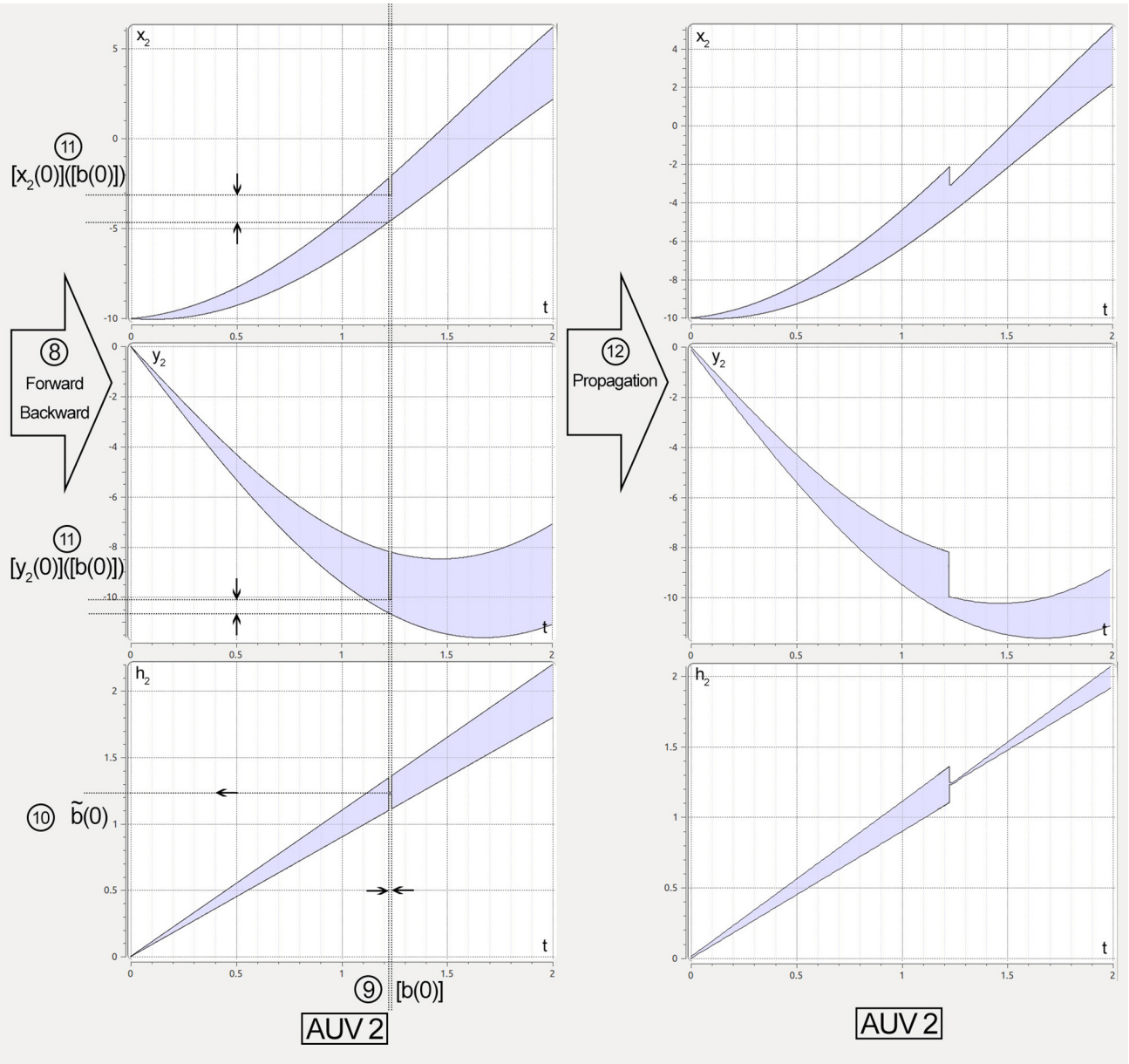

Figure 5.7: (8) We apply our forward-backward algorithm, which contract the intervals $[b(0)]$, $[y2(0)]([b(0)])$ and $[x2(0)]([b(0)])$. (9) We apply the contraction to the tube $[h_2]$ and re-synchronizing the clock. (10) We can compute the receiving time in the robot's own clock. (11) We apply the contractions to the position tubes $[x_2]$ and $[y_2]$. (12) We use the integral contractors to propagate the constraint to the rest of the tube.

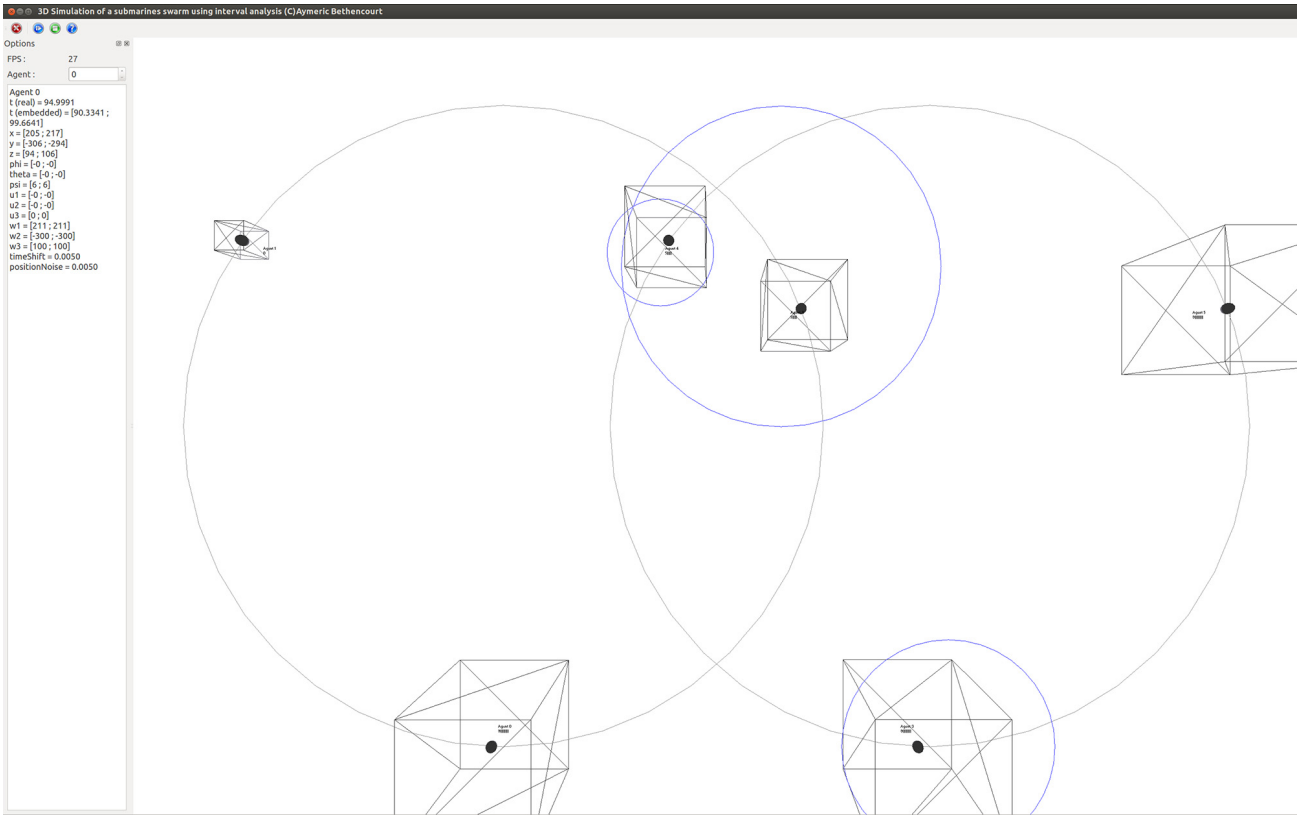

Figure 5.8: SwarmX v2 : 3D simulation of a group of 6 AUVs. The position of the AUVs is represented by the boxes. The red line shows that the robots are in range of communication and the blue circles represent the displacement of the sonar wave. A video of this simulation is available on http://youtu.be/7Uzjr-U7xY4.

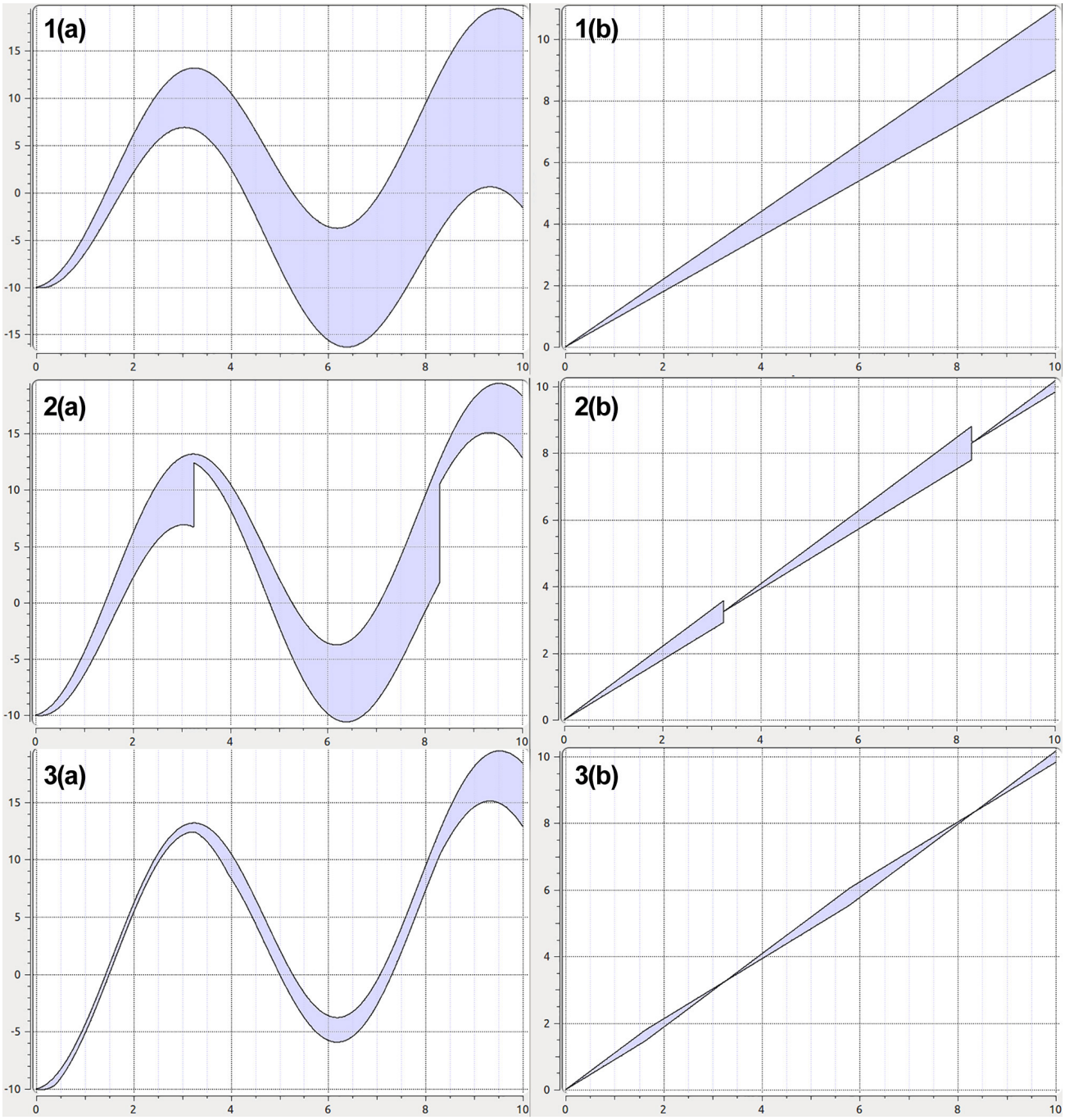

Figure 5.9: Contracted tubes for (a) $[h_4]$ and (b) $[x_4]$ if (1) no ping is received, (2) two pings are received considering online localization, (3) two pings are received considering offline localization.

We illustrated the method with online propagation only, meaning that we contracted the tubes only in the direction of time increasing. However it is possible to consider offline propagation as well, in which we retro-propagate the contractions in decreasing time once the simulation or experiment is done. Fig. 5.9(3) illustrates the principle. Offline localization allows us to contract the tubes of the AUVs with much better accuracy but is only useful once the experiment is over.

This simulation runs in real-time on a 3.2GHz dual core processor.

5.4.3 Sea testing with 2 real AUVs.

Let us now demonstrate the adequacy of our algorithm at sea with the two AUVs from the company *CISCREA* presented in Fig. 5.10. As no regulator was implemented onboard, we had to command the AUVs with a joystick, making them follow approximate circles and reconstruct their trajectories by measuring the command $\mathbf{u}_i$ from the joystick every $0.001s$. AUV1 was staying at the surface so it could continuously use its GPS to contract its position and resynchronize its clock. AUV 2 was supposed to stay underwater (altough it actually stayed on the surface to get the GPS trajectory). The trajectories of the AUVs are represented in fig. 5.10. We then simulated accoustic pings between the AUVs to contract the position and clock of AUV 2 as presented in fig. 5.11.

We notice that the abscissa tube can be contracted. However the clock tube can never be contracted as the noise on the position was too high compared to the clock drift. We indeed chose to use the clock drift of a typical quartz which is 50 ppm (parts per million), which mean that the clock can drift 50 s every 1,000,000 s or approximatly 1 s every 6 hours. This shows the limits of our algorithm as the two first test cases were considering a high clock drift of $0.1s$ per second. In actual AUVs, the clock only drifts much less. Therefore, on small length missions as presented in this experiment, there is no effective contraction on actual clocks drift, except if the clocks are initially drifted for a few seconds before the start of the mission. We could also imagine long term missions of several months, or consider AUVs staying asleep at the bottom of the ocean for several months before awakening with their clock drifted for a few seconds.

5.5 Conclusion

Localizing a group of AUVs while synchronizing their clocks is a difficult problem, mainly due to the inter-temporal constraints on uncertain times. To solve the problem, this chapter has considered the cooperative localization as a constraint satisfaction problem and contracted the boxes around the positions of the AUVs and their clock using a forward-backward algorithm on the inter-temporal measurements made with an acoustic modem. Several test cases were provided, proving the efficiency of the algorithm when the uncertainty on the position and the clock have the same order of magnitude. However we also shown that the algorithm was ineffective at contracting the clock when the orders of magnitude were too far apart, especially on short term missions with accurate clocks.

Finally, we have so far only considered a small group of maximum six AUVs. As the main purpose of this thesis is to study large swarms of robots, the next and final chapter of this thesis will focus on the development of a swarm simulator with hundreds of robots, and show that it is possible to localize them in

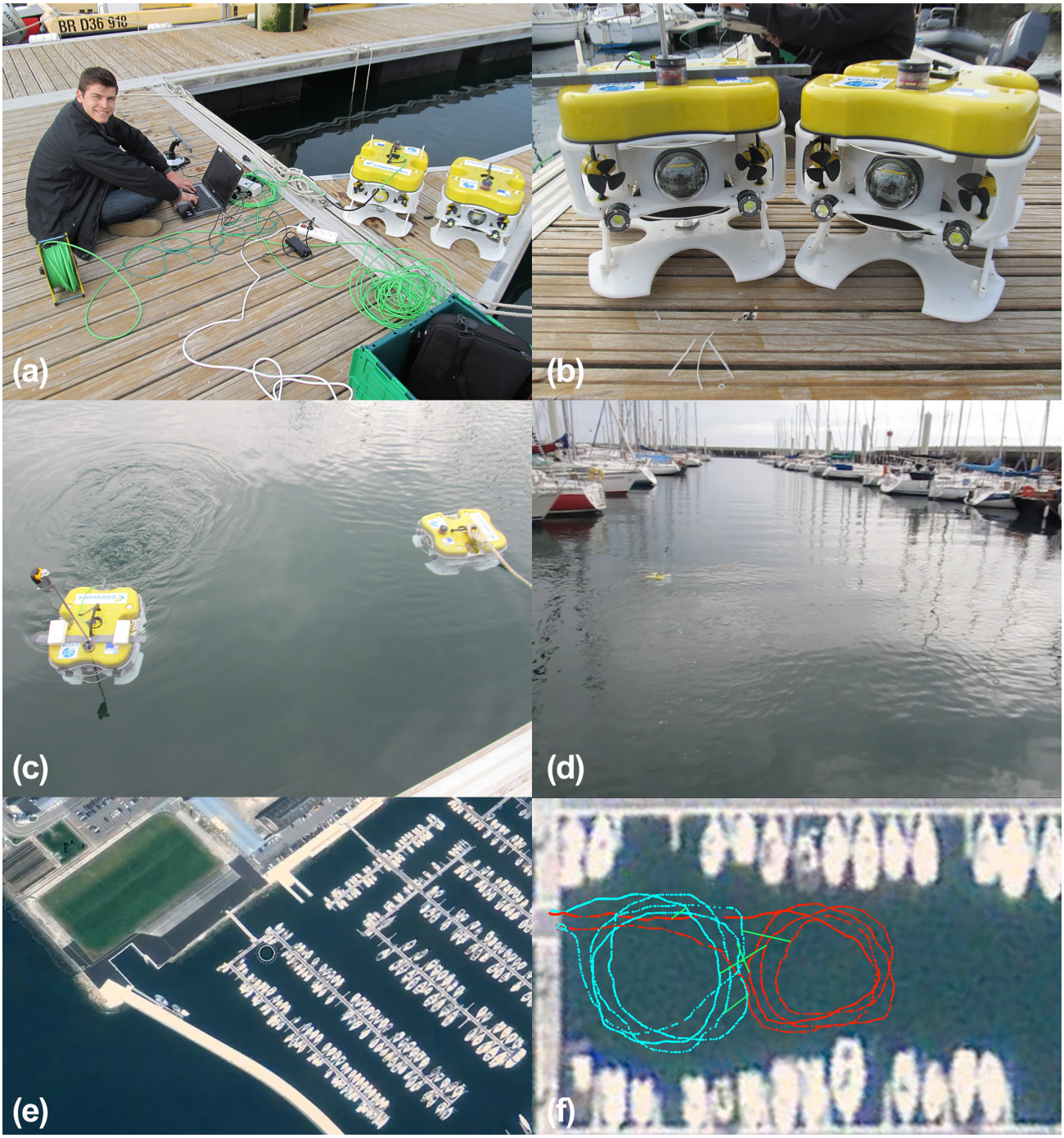

Figure 5.10: (a), (b), (c) and (d) : Sea testing in the Brest harbour, France, with two CISCREA AUVs. (e) Satellite view (© Google Earth) of the testing area. (f) The blue trajectory represents AUV 1 staying on the surface. The red trajectory represents AUV 2 supposed to stay underwater (but actually on the surface as well to collect control GPS data). Both AUVs are commanded with a joystick. The green lines represent simulated sonar pings when the AUV are in range. A video of the experiment is available on http://youtu.be/1QFpko0tYO0.

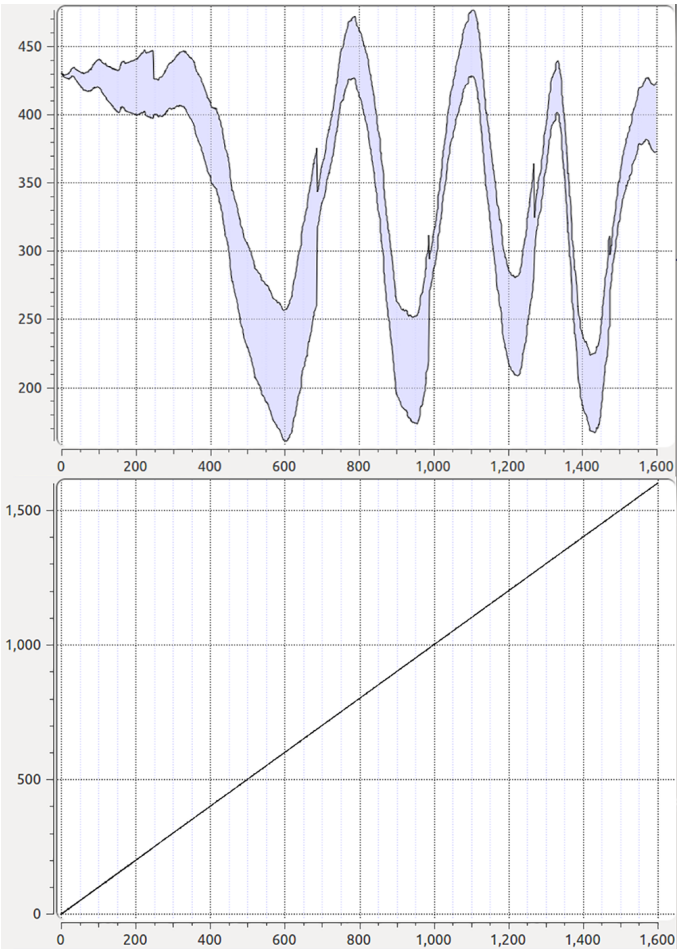

Figure 5.11: Result of contraction on tubes enclosing (a) the abssica x_2 of AUV 2 and (b) its clock h_2.

real-time using the algorithm presented in this chapter.

The work presented in this chapter has been **published** in : *Paladyn, Internatinal Journal of Behaviorhal Robotics, Special issue on underwater robotics, VERSITAS*, 2013.

Chapter 6

Large-scale swarm localization using interval analysis

6.1 Introduction

In the previous chapter, we proposed a distributed model for the localization and clock synchronization of AUVs in a group. Yet the approach failed to demonstrate the scalability and performance of the algorithm. In this chapter, we take an interest in scaling up the demonstration. We introduce a new simulation (that we called *SwarmX v3*) with up to 1,000 simultaneous AUVs running in real-time. For this, we use a collective motion model inspired from flock of birds, herds of animals and schools of fish [Vicsek and Zafeiris, 2012]. These biological systems are spectacular to watch and are based on the same universal pattern: the velocity vectors of neighboring individuals tend to become parallel to each other. The pattern and underlying controlling mechanism seem like a prerequisite to safe, stable and collision free motion. Therefore, it might be advantageous to incorporate the mathematical model that produces this pattern to a group of autonomous robots. As in the previous chapter, each robot is equipped with an acoustic modem and an on-board computer to estimate its state, run our localization algorithm and perform all controlling calculations. This definition prohibits the use of a central processing unit computing the group dynamics like in [Kushleyev et al., 2013] and [Bürkle et al., 2011], but allows the use of external references of positions like the measure of distance to another AUV, or the use of GPS when the AUV surfaces.

According to *Reynolds* [Reynolds, 1987], collective motion can be achieved by following three simple principles : separation (repulsion in short range), cohesion (attraction in long range) and alignment (which aligns the velocity vectors of nearby units). These rules can be written in mathematical form and adapted for the motion of AUVs.

Even the simplest flocking model includes an alignment rule, described as an explicit mathematical axiom [Vicsek et al., 1995], which tends to align the velocity vector of all agents. It is possible to generalize this term by adding coupling of accelerations [Szabó et al., 2009], preferred directions (animal groups) [Couzin et al., 2005] and adaptive decision-making schemes to extend the stability for higher velocities [Dong et al., 2013]. In other models, the alignment rule is a consequence of interaction forces [Grossman et al., 2008]

or velocity terms based on over-damped dynamics like for tissue cells [Szabo et al., 2006].

An important feature of biological flocks is their locality, units adapt each other only in a finite neighborhood. In autonomous robotics, the communication between robots also have a finite range. In other words, units can send messages (e.g. their position, velocity and clock) only to nearby units. In our example, AUVs will be limited by the range of their acoustic modem. Another analogy between biological flocks and robotic swarms is that they are both agent-based. Each bird in a flock evolves individually through the dynamic system, and likewise each AUV in our model has its own computer and on-board sensors to control its individual dynamic.

Because of these similarities, some properties of flocking models can be integrated into the control dynamics of autonomous robots. There is a wide range of methods using features of collective behavior in robotics [Brambilla et al., 2013][Turgut et al., 2008][Hauert et al., 2011]. Yet none of them consider using interval analysis to localize individuals in a swarm. We believe that interval analysis can easily be applied to swarming models and perform well, especially since interval methods are well known to work best when the number of equations is far superior to the number of parameters to evaluate.

The goal of this chapter is to demonstrate that interval analysis can be applied to swarming models and that the algorithm presented in Chapter 5 can be applied to a large swarm of hundreds of AUVs simultaneously. Section 2 recalls our AUV model with a few modifications and section 3 presents *Reynolds*' rules and how to adapt them to our model. Section 4 presents the simulated results for 300 and 1,000 AUVs and Section 5 concludes the chapter.

6.2 AUV model

In this section, we recall the AUV model used in this Chapter 5 and based on a realistic robotic system.

For a robot $i \in \{1, \ldots, m\}$ in a swarm and acoustic ping $k \in \{1, \ldots, k_{\max}\}$, we consider the state equations of the robot as follows:

$$\begin{array}{lll}
\text{(i)} & \dot{\mathbf{x}}_i(t) = \mathbf{f}(\mathbf{x}_i(t), \mathbf{u}_i(t)) + \mathbf{n}_{\mathbf{x}}(t) & \text{(Evolution equation)} \\
\text{(ii)} & g\left(\mathbf{x}_{i(k)}(a(k)), \mathbf{x}_{j(k)}(b(k)), a(k), b(k)\right) = 0 & \text{(Ping observation equation)} \\
\text{(iii)} & \tilde{a}(k) = h_{i(k)}(a(k)) & \text{(Local time function of AUV } i\text{)} \\
\text{(iv)} & \tilde{b}(k) = h_{j(k)}(b(k)) & \text{(Local time function of AUV } j\text{)} \\
\text{(v)} & \dot{h}_i(t) = 1 + n_h(t) & \text{(Clock function of AUV } i\text{)}
\end{array} \tag{6.1}$$

(i) corresponds to the state equations of the i-th robots, where $\mathbf{x}_i$ is the state vector of the robot i, $\mathbf{u}_i$ its inputs or commands, $\mathbf{y}_i$ its outputs or measures. The state noise $\mathbf{n}_{\mathbf{x}}(t)$ is assumed to be bounded.

(ii) is the inter-temporal observation function where $a \in \mathbb{R}$ corresponds to the emission time, $b \in \mathbb{R}$ to the reception time, $i \in \{1, \ldots, m\}$ the emitting robot, and $j \in \{1, \ldots, m\}$ the receiver. Here we have $g : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}$ so that:

$$g(\mathbf{x}_1, \mathbf{x}_2, a, b) = \|\mathbf{x}_1 - \mathbf{x}_2\| - c * (b - a) \tag{6.2}$$

where c is the speed of sound. Using the Euclidean norm, (ii) can be re-written as

$$c * (b(k) - a(k)) = \sqrt{\begin{array}{c}(x_{i(k)}(a(k)) - x_{j(k)}(b(k)))^2 \\ +(y_{i(k)}(a(k)) - y_{j(k)}(b(k)))^2\end{array}} \tag{6.3}$$

(iii) $\tilde{a}(k)$ corresponds to the local time of the robot $i(k)$ (i.e. the emitter has emitted the k^{th} ping).

(iv) $\tilde{b}(k)$ corresponds to the local time of the robot $j(k)$ (i.e. the receiver has received the k^{th} ping).

(v) We denote by $\tau = h_i(t)$ the clock function [Srikanth and Toueg, 1987] which for an absolute time t matches the inner time τ of the robot i. $n_h(t)$ is the clock noise and is assumed to be bounded.

Notice that for all k, we know exactly: $\tilde{a}(k), \tilde{b}(k), i(k), j(k)$.

Each acoustic ping encloses some data: The position box and emission interval of the emitting AUV as described in Chapter 5. In this chapter however pings will also enclose the box around the velocity vector of the AUV as this will be useful for the swarming behaviors presented in the next section. Basically, each ping k transmits $[\mathbf{x}_{i(k)}(a(k))]$, $[\dot{\mathbf{x}}_{i(k)}(a(k))]$ and $[\tilde{a}(k)]$ to all AUVs that are in range.

We use a controlling algorithm with an input, the preferred velocity vector of the AUV. During the flocking movements, the time-dependance of the preferred velocity of the $i-$th AUV can be a function of the position and velocity of the other AUVs :

$$\dot{\mathbf{x}}_i^{preferred}(t) = \mathbf{e}^i(\{\mathbf{x}_j(t)\}_{j=1}^N, \{\dot{\mathbf{x}}_j(t)\}_{j=1}^N, t) \tag{6.4}$$

where N will be the number of j nearby (in range) neighbors, and e^i is the control function enclosing arbitrary features of the controlling dynamics. Note that the controller algorithm can be e.g. a PID controller. The acceleration of the unit is of course limited by the power and inertia parameters.

6.3 Behavioral command

6.3.1 Reynolds' rules

The controlling algorithm is based on the three basic principles first proposed by *Craig Reynolds* in his ground breaking 1987 paper [Reynolds, 1987]. The principles (also known as *Reynolds' rules*) propose three steering behaviors: *separation*, *cohesion* and *alignment*. Each principle determines how a member of the swarm reacts to other members in its local *neighborhood*. Members of the swarm that are not in its local neighborhood are ignored. The neighborhood is specified by a distance d which defines when two members are "nearby", or in our case in range of acoustic communication.

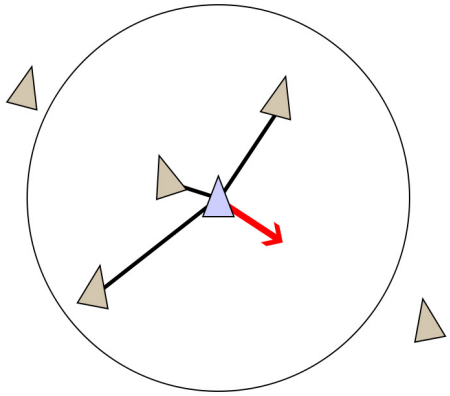

Figure 6.1: Illustration of the separation rule.

	Algorithm *SEPARATION* (in : *AUV*, out : **dvelocity**)
1	**dposition** := **0**; **dvelocity** :=**0**
2	*neighbours* := getNeighbours(*AUV*)
3	foreach *nAUV* in *neighbours* do
4	**dposition**+ = *nAUV*.**position** − *AUV*.**position**
5	end
6	**dposition** = **dposition**/*neighbours.size*()
7	**dvelocity** = −**dposition**/*dt*
8	return **dvelocity**

Table 6.1: SEPARATION Algorithm

6.3.2 Separation

Separation is a steering behavior that gives a member of the swarm (here an AUV) the ability to maintain a certain distance from the others nearby AUVs. This allows AUVs to avoid collisions. To compute steering for separation, the AUV first searches for other AUVs in its neighborhood. For each nearby AUV, a repulsive force is computed by subtracting the position of the AUV to the nearby AUVs , normalizing it and then applying an $1/d$ weighting. The repulsive forces from each nearby AUV are summed up to produce the overall steering force. Fig. 6.1 illustrates the rule and Table 6.1 presents the algorithm.

6.3.3 Cohesion

Cohesion is the second steering behavior and gives an AUV the ability to cohere with (approach and form a group with) other nearby AUVs. For this, the AUV first searches for other AUVs in its neighborhood, as in Separation. Then it computes the average position, or center of gravity, and applies a steering force in that direction. Cohesion and Separation are usually in opposition and insure that an AUV is never too far nor too close to nearby AUVs. Fig. 6.2 illustrates the rule and Table 6.2 presents the algorithm.

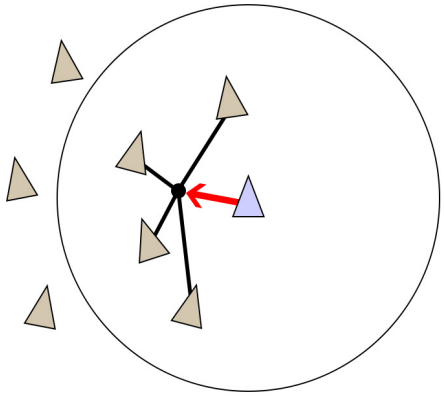

Figure 6.2: Illustration of the cohesion rule.

	Algorithm *COHESION* (in : *AUV*, out : **dvelocity**)
1	**dposition** := **0**; **dvelocity** :=**0**
2	*neighbours* := getNeighbours(*AUV*)
3	foreach *nAUV* in *neighbours* do
4	**dposition**+ = *nAUV*.**position**
5	end
6	**dposition** = **dposition**/*neighbours.size*()
7	**dvelocity** = (**dposition** − *AUV*.**position**)/*dt*
8	return **dvelocity**

Table 6.2: COHESION Algorithm

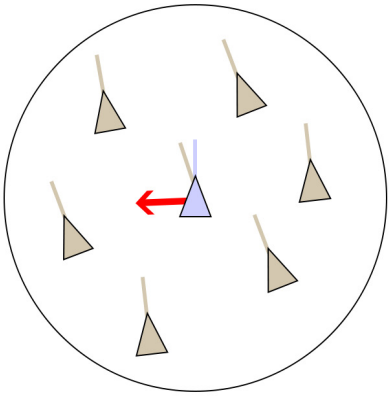

Figure 6.3: Illustration of the alignment rule.

	Algorithm *ALIGNMENT* (in : *AUV*, out : **dvelocity**)
1	**dvelocity** := **0**
2	*neighbours* := getNeighbours(*AUV*)
3	foreach *nAUV* in *neighbours* do
4	**dvelocity**+ = *nAUV*.**velocity**
5	end
6	**dvelocity** = **dvelocity**/*neighbours.size*()
7	return **dvelocity**

Table 6.3: ALIGNMENT Algorithm

6.3.4 Alignment

Alignment is the last steering behavior and gives an AUV the ability to align itself with nearby AUVs. Steering for alignment can be computed by averaging the velocity vector of all nearby AUVs and subtracting this value to the current velocity vector of the AUV. This steering will tend to align the AUV with its neighbors. Fig. 6.3 illustrates this rule and Table 6.3 presents the algorithm.

6.3.5 Swarming

The *swarming (or flocking) behavior* is the addition of the separation, cohesion and alignment behaviors, which together form a motion very similar to what is found in animal flocks, herds and schools. Table 6.4 presents the swarming algorithm, in which we first normalized the three steering components for better control. Note that in some applications [Hodgins and Brogan, 1994][Tu and Terzopoulos, 1994][Tu, 1999], a weighting factor can be applied to each behavior to better balance the system.

	Algorithm $SWARMING$ (inout : AUV)
1	**alignment** := $ALIGNMENT(AUV).normalize(1)$
2	**cohesion** := $COHESION(AUV).normalize(1)$
3	**separation** := $SEPARATION(AUV).normalize(1)$
4	AUV.**velocity**.$x+=$ **alignment**.x + **cohesion**.x + **separation**.x
5	AUV.**velocity**.$y+=$ **alignment**.y + **cohesion**.y + **separation**.y
6	AUV.**velocity**.$z+=$ **alignment**.z + **cohesion**.z + **separation**.z

Table 6.4: SWARMING Algorithm

6.3.6 Adapting the Reynold's rules to our model

Reynolds' rules can easily be adapted to our AUV model. The neighborhood distance to consider is the range of the acoustic of the AUVs. Each AUV emits a acoustic ping every second which encloses the position box, velocity box and emitting time interval of the emitting robot. We consider the center of each box as the position and velocity that we provide to the computation of the *Reynolds' rules*.

As the rules are based on finding neighbors at the same given time, our algorithm has to estimate the position of all neighbors by predicting their position from the latest acoustic ping received from the AUV. As the pings are emitted every second and enclose the position and velocity of the AUV, it is possible to estimate the position with a good accuracy. We then run the forward-backward algorithm presented in Chapter 5 to contract the position box and clock interval of the AUV. All interval computations are handled by *IBEX*, that we integrated directly within our *SwarmX v3* simulation.

Notice that the separation rule is usually defined by a static parameter, which is fine when the position of the AUV is precisely known. In our case, AUVs do not know their real position but know a position box, in which they are guaranteed to be. When the width of the boxes increases or decreases, the separation distance must be adapted in real time to guarantee that no collision will happen. The minimum separation distance between a robot i and j is therefore :

$$d^{separation} = \frac{width([\mathbf{x}_i(t)]) + width([\mathbf{x}_j(t)])}{2} \tag{6.5}$$

6.4 Test cases

6.4.1 With 3 AUVs

The simulation takes place in a $1km^3$ underwater environment of spatial coordinates $(x, y, z) = [0, 1000] \times [0, 1000] \times [-1000, 0]$. Each AUV starts with a random position box, that will inflate over time as state noise occurs. We consider that they can use the GPS to contract their position box and clock interval when they get near the surface ($z > -100$). Each AUV emits a acoustic ping every second enclosing the position box, velocity box and clock interval. If another AUV receives the ping, it will apply the localization algorithm presented in Chapter 5 to contract its own position box and clock interval. The swarming algorithm is

applied for every sampling time of the simulation.

Fig. 6.4 presents a simple simulation of 3 AUVs. (a) shows the position of the AUVs. (b) displays vertices when AUVs are in range of communication, but does not mean that a ping have actually been emitted. (d) represents the actual acoustic pings. Finally, (c) shows the position box around each AUV. When an AUV is too deep to use the GPS and is not in range of acoustic communication with another AUV, these boxes keep inflating with the state noise. As the box inflate, the preferred distance for the separation rule also increases (as presented in 6.5) to guarantee that the boxes never intersect and therefore that the AUVs never collide. In this example, the AUVs keep on moving away from each other until one of them can access the GPS and transmit its contracted box to the other two AUVs.

Keep in mind that although the simulation runs on a single computer, the algorithm is completely distributed. Each AUV is coded as an autonomous individual that is only aware of itself and its neighborhood.

6.4.2 With 300 AUVs

In this simulation, we spawned 300 AUVs at $t_0 = 0s$ at random locations (Fig. 6.5(a)) with random starting position boxes (Fig. 6.5(c)). We then run the simulation for $20s$. Fig. 6.6 presents the result. We can clearly observe that the *Reynolds' cohesion rule* makes all the AUVs gather into groups, but the separation rule still prevents any collision. We also notice that the AUVs in groups have nearly the same direction, result of the alignment rule. AUVs on top of the box are close to the surface and can contract their position box using the GPS. A few pings are then sufficient to propagate the contractions to the bottom of the swarm which then maintain a good localization through the whole swarm. Fig. 6.7 represents the tubes within IBEX for the abscissa and ordinate of one of the bottom AUVs.

6.4.3 With 1,000 AUVs

We now introduce 1,000 AUVs into the simulation. To increase the difficulty, we also consider three distinct groups of AUVs. The green AUVs are faster than the default blue AUVs. They are considered cheaper version, and therefore are smaller and equipped with small-range acoustic modems. The red AUVs are bigger than the blue AUVs and are more expensive and slow due to their inertia. They are equipped with long-range acoustic modem. The three groups are communicating for collision avoidance (separation) but do not trust each other for other tasks like alignment and cohesion for unclear reasons (e.g. they belong to different companies). The simulation presented in 6.8 shows that small groups are formed by the green swarm. The blue swarm usually try to regroup but is often broken apart by the red swarm that rarely form cohesive groups. We can also notice that the more AUVs we add to the simulation, the better the localization and clock synchronization, but the less cohesive are the AUVs.

6.4.4 Performances

All the simulations presented in this chapter are running in real-time at minimum 30 FPS on a 3.2GHz Intel core i7. Frame lag begins to appear above 1,000 AUVs. Keep in mind that this simulation is running on

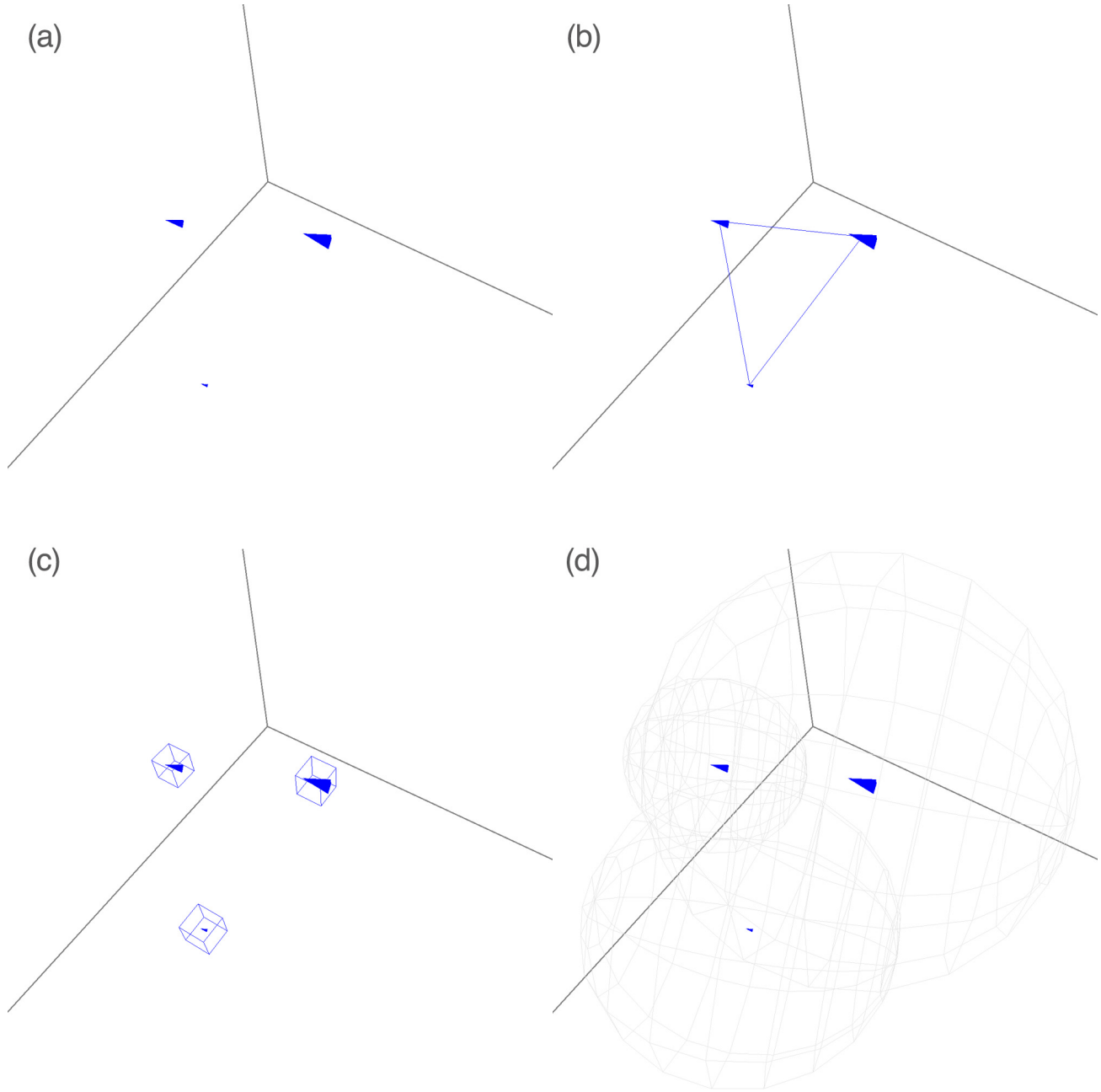

Figure 6.4: SwarmX v3 : (a) Position of the AUVs. (b) Vertices appears between the AUVs when they are in range of communication. (c) Localization boxes of the AUVs. (d) Sonar pings emitted by the AUVs.

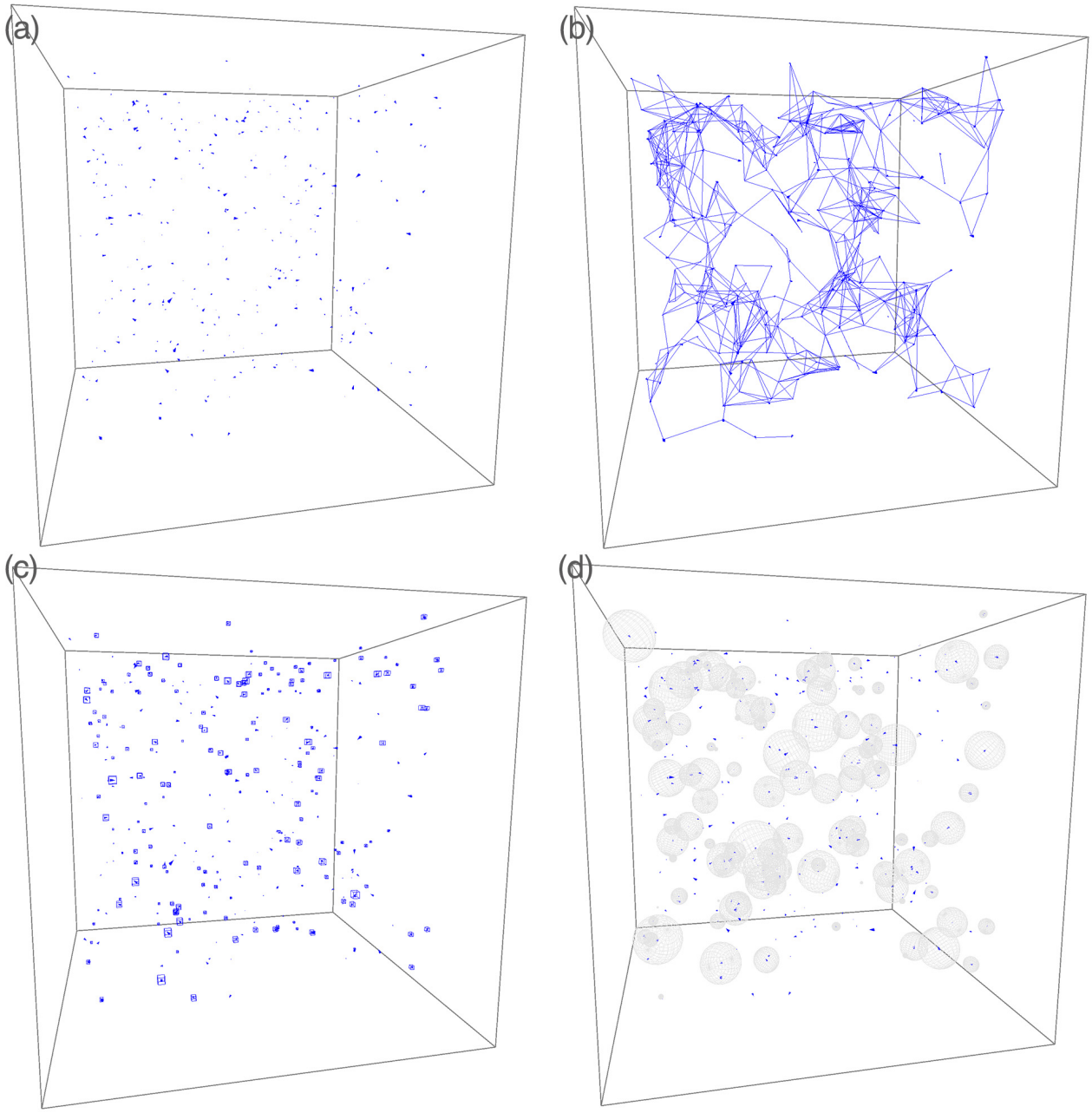

Figure 6.5: At $t_0 = 0s$ we randomly introduce 300 AUVs into the simulation.

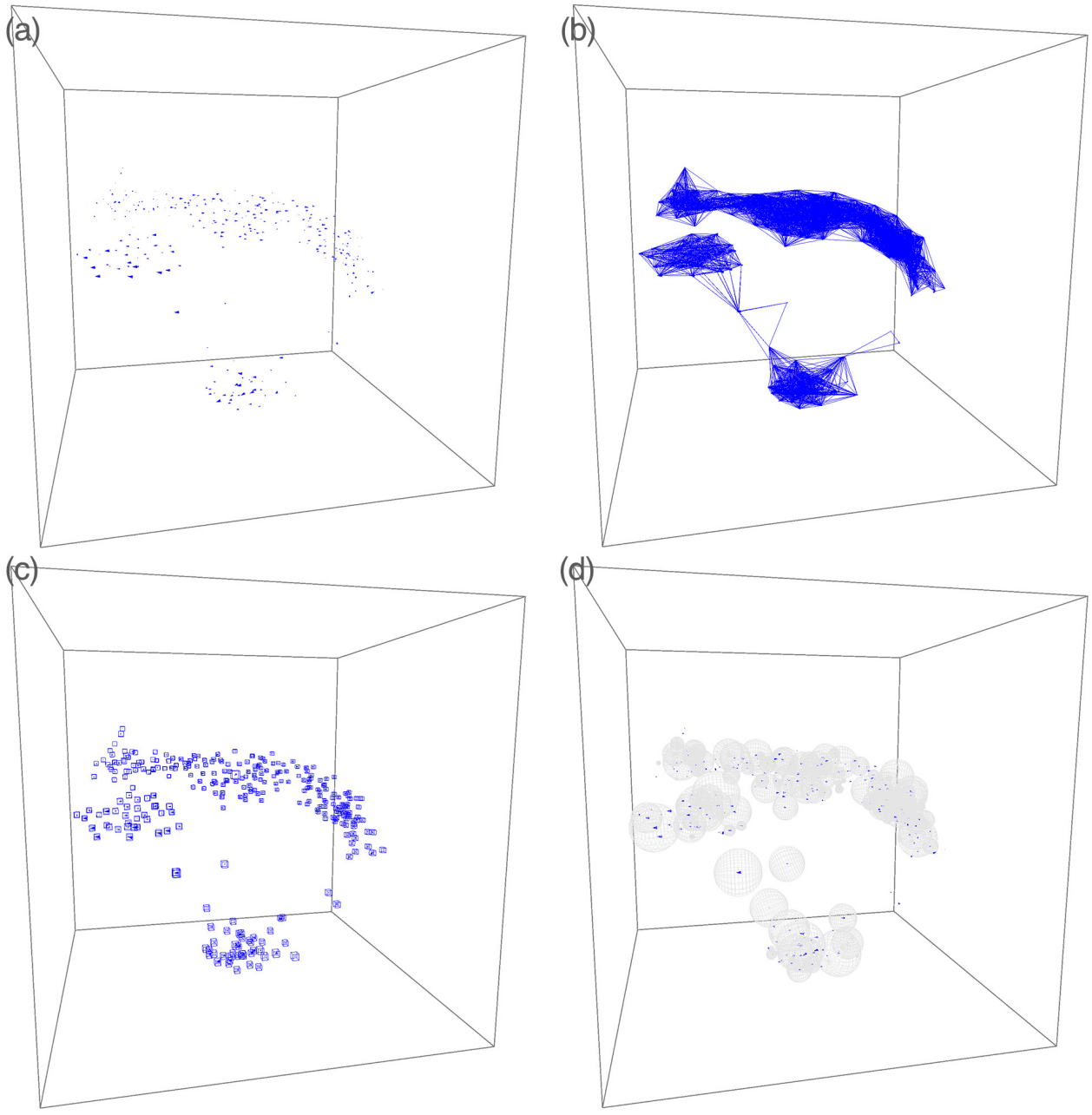

Figure 6.6: At $t_f = 30s$, all AUVs are in swarming positions and well localized. A video of the simulation is available on http://youtu.be/F4ntGSBS1J4.

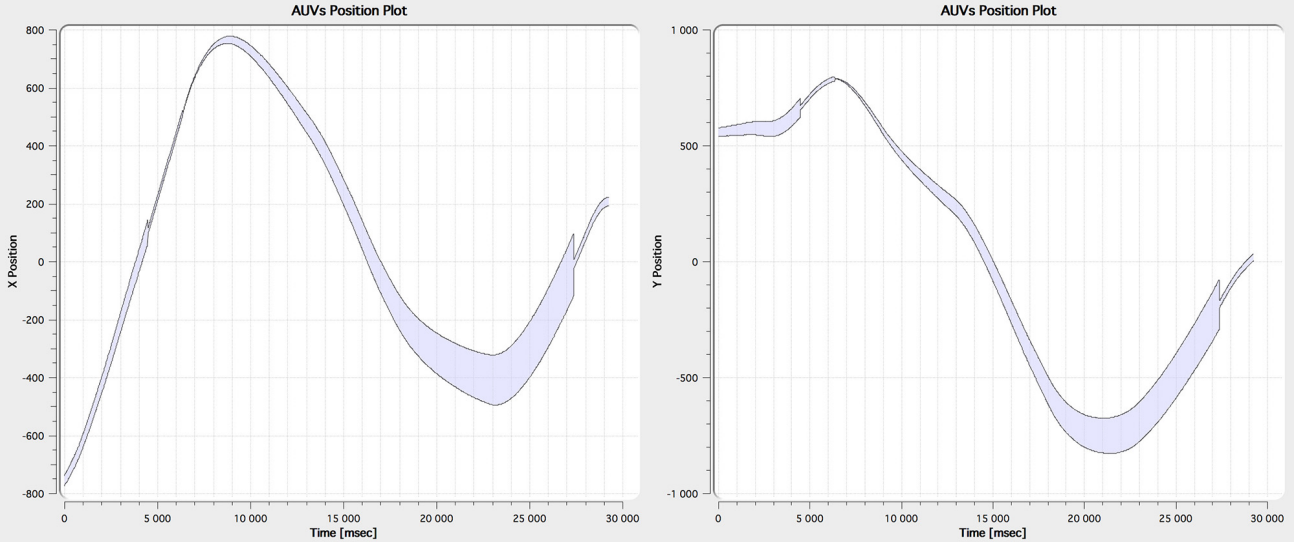

Figure 6.7: Tubes $[x_i]$ and $[y_i]$ positions from $t_0 = 0s$ to $t_f = 30s$ for a random AUV i.

a single computer but that the localization algorithm is in reality distributed and running on each AUV, which demonstrates the speed and scalability of the algorithm.

6.5 Conclusion

In this chapter, we showed that it was possible to scale up our localization algorithm while maintaining good performances. As we did in Chapter 5, the simulation could be considered offline and retro-propagate the contractions on the tubes for a better localization after the mission. As always, a video of the simulation and the source code is available on http://aymericbethencourt.com/thesis/.

The work presented in this chapter has been **submitted** to : *Swarm intelligence*, 2014, and is currently awaiting reviews.

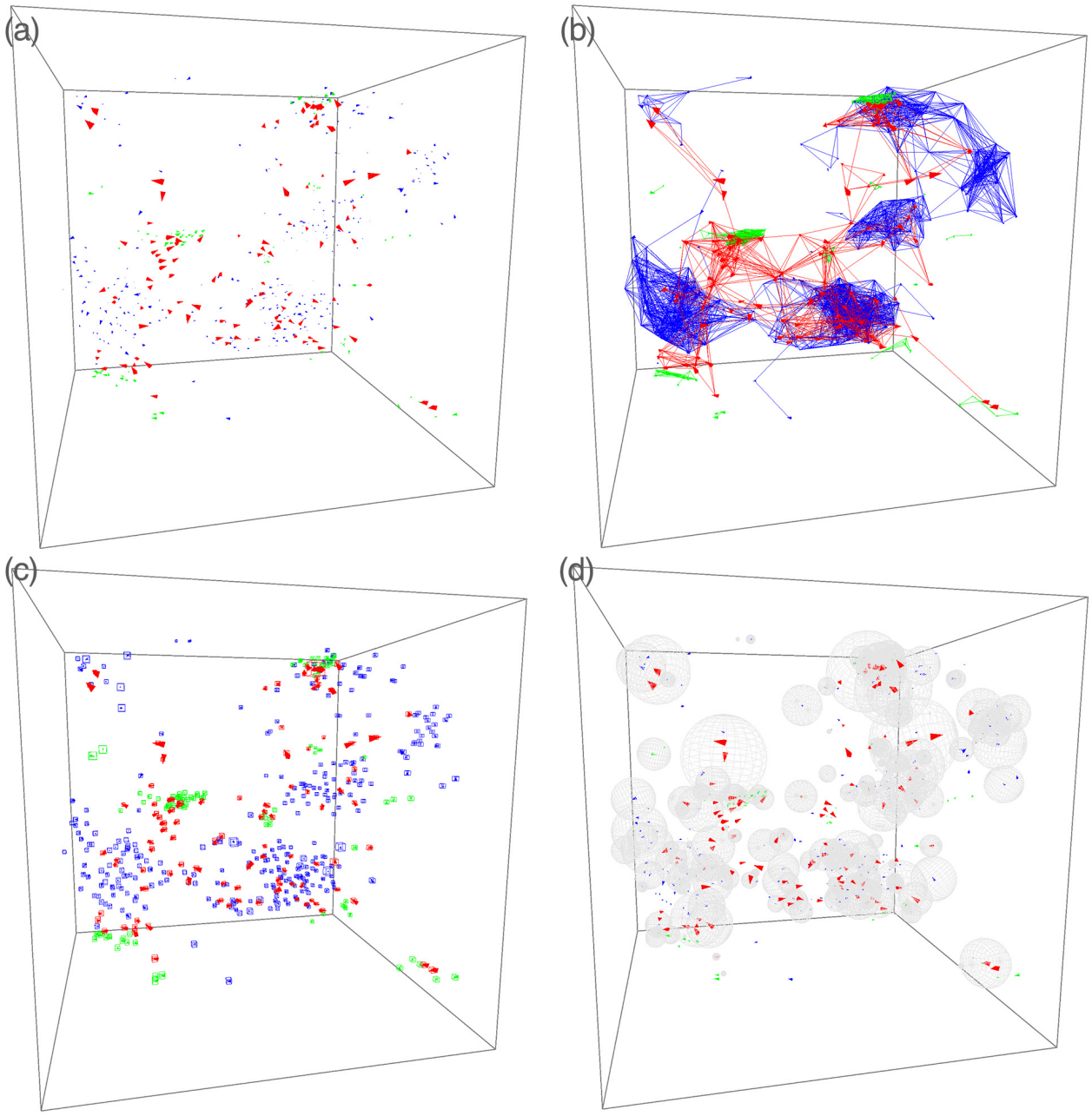

Figure 6.8: Simulation with 1,000 AUVs of 3 types.

Chapter 7

Conclusion

In this thesis we proposed a new way to localize a swarm of AUVs using interval analysis.

In the first chapter we introduced the notion of localization and why it is important for autonomous robots, especially underwater.

In the second chapter we presented the basic notions of interval arithmetic and a few state-of-the-art algorithms to estimate parameters of a given system with observations of this system with bounded errors. Using set inversion via interval analysis, we were able to compute an inner and outer approximation of a simple range-only localization problem with outliers.

In the third chapter we extended contractor algebra to allow for the geometrical transformation of contractors and showed that it was possible to build minimal contractors in a very easy way for some constraints with symmetries. As an application, we considered the construction of a contractor associated with the constraint $\theta = atan2\,(x, y)$ using central, axial and 2π-modulo symmetries, and showed that this contractor was minimal. We integrated the contractor in IBEX and demonstrated its efficiency on a angle-only localization problem.

In the fourth chapter we no longer considered static problems but modelized dynamic systems. We introduced the notion of *tubes*, or *interval of functions* that encloses intervals at different times and encompasses the informations needed to guarantee associations upon trajectories. Then, an arithmetic was developed around this notion, and a contractor-based approach followed. As a result, a method was proposed to contract tubes that enclose the solution. Several test cases were provided to demonstrate the approach, including the estimation of the state of AUVs. We programmed tubes and their contractors within IBEX, and created a simulator, namely *SwarmX v1*, to simulate a group of 6 AUVs able to communicate underwater to exchange localization data. The inter-temporal constraints were solved using tubes.

In the fifth chapter we pushed the simulation further by considering that the clocks of the robots were unsynchronized, making the time measurements uncertain. To solve this problem, we considered cooperative localization as a constraint satisfaction problem and contracted the boxes around the positions of the AUVs and their clock using a forward-backward algorithm on inter-temporal measurements made possible by an acoustic modem equipping each AUV. *SwarmX v2* was developed to prove the efficiency of the algorithm when the uncertainty on the position and on the clock drift had the same order of magnitude. However real

tests at sea showed that the algorithm was inefficient at contracting the clock when the orders of magnitude were too far apart, especially on short term missions with accurate clocks.

Finally, the sixth chapter focused on studying the scalability of such approach. We created *SwarmX v3* to simulate 1,000 AUVs evolving in real-time and following the three *Reynolds' rules*.

Further work can include the study of outliers, integration with DAEs, coupling with probabilistic methods and integration with guaranteed differential evaluation methods.

Videos, sources codes and pdf versions of the published articles are available on http://aymericbethencourt.com/thesis/.

Appendix A

Visual localization and 3D reconstruction using the Kinect device coupled with an IMU.

As the final project of my master degree, I had my first experience with interval analysis using the *Kinect* device to reconstruct scenes and objects in 3D. As the subject is relevant to this thesis, this appendix presents my work.

A.1 Introduction

The *Microsoft Kinect* sensor device was released for the Microsoft Xbox 360 video game console at the end of the year 2010. The device allowed a user to play video games just by moving his body and therefore allowed gaming without the use of any game pad or joystick. The *Kinect* includes a color RGB camera, an infrared depth sensor, an accelerometer, four microphones and a motor to adjust the tilt. In addition to the commercial success of the *Kinect* as a gaming device, it attracted a lot of interest from the scientific community thanks to its numerous integrated features, its low price and its shelf availability. The depth sensor is in fact a near-infrared projector that projects a known structured pattern of speckles that is being observed by a CMOS IR camera. Each speckle is unique and can be recognize from others. The device then computes the triangulation of each speckle between the known virtual pattern and the observed pattern to construct the depth image. The calibration between the projector and the camera has of course to be known. The depth images can be represented as a 3D metric points cloud by projecting the image points into the real world coordinate:

$$\overrightarrow{x} = K[R|t]\overrightarrow{X} \tag{A.1}$$

where:

$\overrightarrow{x}$ is the homogeneous coordinate vector of a point in the image,

K is the intrinsic parameters matrix (available in the datasheets),

R and t are the extrinsic parameters matrices (respectively equal to identity and 0 since we do not consider any rotation or translation here),

$\overrightarrow{X}$ is the homogeneous coordinate vector of a point in the world.

By expending the matrices, we obtain:

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \begin{pmatrix} f_x X + c_x Z \\ f_y Y + c_y Z \\ Z \end{pmatrix} \tag{A.2}$$

where:

X, Y, Z are the homogeneous world coordinates of a point,

x, y, w are the homogeneous image coordinate of the same point,

f_x, f_y are the focal length on each direction,

c_x, c_y are the coordinates of the principal point of the camera.

Since we are in homogeneous coordinates, we can write the inverse relation for X and Y as follows:

$$X = \frac{(x - c_x)Z}{f_x} \text{ and } Y = \frac{(y - c_y)Z}{f_y} \tag{A.3}$$

Notice that the particularity of this type of camera is that we know the depth Z. The device also has a RGB camera which needs to be calibrated in order to associate a color to a depth pixel. For that, we have to use the intrinsic parameters of both cameras and the extrinsic mapping between the two cameras [**?**]. The mapping can be expressed as the following:

$$\begin{pmatrix} X_{rgb} \\ Y_{rgb} \\ Z_{rgb} \end{pmatrix} = R \begin{pmatrix} X_{ir} \\ Y_{ir} \\ Z_{ir} \end{pmatrix} + t \tag{A.4}$$

where:

$X_{rgb}, Y_{rgb}, Z_{rgb}$ are the homogeneous coordinates of a point in the rgb camera frame,

X_{ir}, Y_{ir}, Z_{ir} are the homogeneous image coordinates of a point in the ir sensor frame,

R and t represent the transformation between the rgb camera and ir sensor.

The RGB and depth images can therefore be represented together as a colored metric points cloud. In section 2, we examine the state of the art for computing the transformation between two points cloud. We then present a new approach using interval analysis in section 3 and add an IMU to the *Kinect* to optimize the performances in section 4.

Figure A.1: Kinect's projected IR structured light

A.2 Standard algorithms

A.2.1 Principle

We tested most of the existing open source methods including:

- *RGB-D Mapper* by P. Henry, M. Krainin , E. Herbst, X. Ren and D. Fox [Henry et al., 2010].

- *RGBDemo* by N. Burrus.

- *RGB-D SLAM* by N. Engelhard, F. Endres, J. Hess, J. Sturm , W. Burgard [Engelharda et al., 2011].

- *KinectFusion* by A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim and A. Fitzgibbon [Izadi et al., 2011].

They all more or less use the same algorithms based on 4 steps: First, they extract *SIFT* features (or its variants like *SURF*) from the incoming color images. Then they match these features against features from the previous images. By evaluating the depth images at the locations of these feature points, they obtain a set of point-wise 3D correspondences between any two frames. Based on these correspondences, they estimate the relative transformation between the frames using *RANSAC*. The third step is to improve this initial estimate using a variant of the *ICP* algorithm. As the pair-wise pose estimates between frames are not necessarily globally consistent, they optimize the resulting pose graph in the fourth step using a pose graph solver like *HOG-Man*. The output of their algorithm is a globally consistent 3D model of the perceived environment, represented as a colored point cloud.

A.2.2 About Sift

Scale-Invariant Feature Transform (or *SIFT*) [Lowe, 1999] is an algorithm in computer vision to detect and describe local features in images. The algorithm was published by *David Lowe* in 1999. For any object in

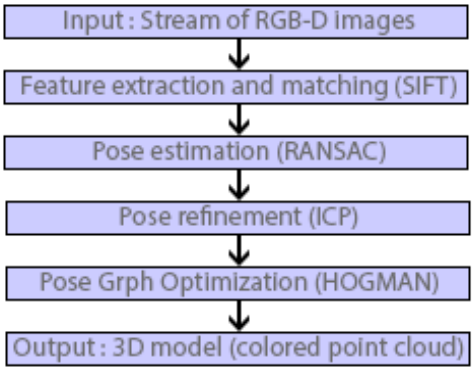

Figure A.2: Principle of existing methods

an image, interesting points on the object can be extracted to provide a "feature description" of the object. This description, extracted from a training image, can then be used to identify the object when attempting to locate it in a test image containing many other objects. To perform reliable recognition, it is important that the features extracted from the training image be detectable even under changes in image scale, noise and illumination. Such points usually lie on high-contrast regions of the image, such as object edges.

The key stages in the SIFT algorithm are:

- Scale-invariant feature detection: *Lowe*'s method for image feature generation transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion.

- Feature matching and indexing: Indexing consists of storing the feature vectors and identifying matching feature vectors from the new image. *Lowe* used a modification of the k-d tree algorithm called the Best-bin-first search method that can identify the nearest neighbors with high probability using only a limited amount of computation.

- Cluster identification by *Hough Transform* voting: *Hough Transform* is used to cluster reliable model hypotheses to search for feature vectors that agree upon a particular model pose. *Hough Transform* identifies clusters of features with a consistent interpretation by using each feature to vote for all object poses that are consistent with the feature. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature. An entry in a hash table is created predicting the model location, orientation, and scale from the match hypothesis. The hash table is searched to identify all clusters of at least 3 entries in a bin, and the bins are sorted into decreasing order of size.

A.2.3 About SURF

Speeded Up Robust Feature (or *SURF*) [Bay et al., 2006] is also a robust image detector & descriptor, first presented by *Herbert Bay* in 2006. It is partly inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image

transformations than SIFT. SURF is based on sums of approximated 2D *Haar wavelet* responses and makes an efficient use of integral images. It uses an integer approximation to the determinant of *Hessian* blob detector, which can be computed extremely quickly with an integral image. For features, it uses the sum of the *Haar wavelet* response around the point of interest.

A.2.4 About RANSAC

RANSAC is an abbreviation for *RANdom SAmple Consensus* [Fischler and Bolles, 1987]. It is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The algorithm was first published by *Fischler and Bolles* in 1981. A basic assumption is that the data consists of "inliers", i.e., data whose distribution can be explained by some set of model parameters, and "outliers" which are data that do not fit the model. In addition to this, the data can be subject to noise. The outliers can come, e.g., from extreme values of the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data. RANSAC also assumes that, given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data. RANSAC achieves its goal by iteratively selecting a random subset of the original data.

These data are hypothetical inliers and this hypothesis is then tested as follows:

- A model is fitted to the hypothetical inliers, i.e. all free parameters of the model are reconstructed from the inliers.

- All other data are then tested against the fitted model and, if a point fits well to the estimated model, also considered as a hypothetical inlier.

- The estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers.

- The model is reestimated from all hypothetical inliers, because it has only been estimated from the initial set of hypothetical inliers.

Finally, the model is evaluated by estimating the error of the inliers relative to the model. This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model. An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set. A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters. When the number of iterations computed is limited, the solution obtained may not be optimal, and it may not even be one that fits the data in a good way. In this way RANSAC offers a trade-off; by computing a greater number of iterations the probability of a reasonable model being produced is increased. Another disadvantage of RANSAC is that it requires the setting of problem-specific thresholds.

A.2.5 About ICP

Iterative Closest Point (*ICP*) [Besl and McKay, 1992][Rusinkiewicz and Levoy, 2001] is an algorithm employed to minimize the difference between two clouds of points. ICP is often used to reconstruct 2D or 3D surfaces from different scans, to localize robots and achieve optimal path planning [Yang and Medioni, 1992] (especially when wheel odometry is unreliable due to slippery terrain), to co-register bone models, etc. The algorithm is conceptually simple and is commonly used in real- time. It iteratively revises the transformation (translation, rotation) needed to minimize the distance between the points of two raw scans. The inputs are points from two raw scans, initial estimation of the transformation, criteria for stopping the iteration, and the output is the refined transformation.

Essentially, the algorithm steps are:

- Associate points by the nearest neighbor criteria.

- Estimate transformation parameters using a mean square cost function.

- Transform the points using the estimated parameters.

- Iterate (re-associate the points and so on).

A.2.6 About HOG-Man

HOG-Man [Grisetti et al., 2010] is an optimization approach for graph-based SLAM (Simultaneous localization And Mapping). It provides a highly efficient error minimization procedure that considers the underlying space is a manifold and not an Euclidian space. It furthermore generates a hierarchy of pose-graphs which is used perform the operations during online mapping in a highly efficient way.

A.3 Our method

We chose to keep the principle of finding the correspondences between the two 2D images then use them to compute the transformation between the 3D points clouds. However, instead of using SIFT, we chose to use A-SIFT, and instead of using probabilistic methods like RANSAC, we developed our own algorithm based on interval analysis techniques.

A.3.1 About A-SIFT

While SIFT is fully invariant with respect to only three parameters namely zoom, rotation and translation, the new method treats the two remaining parameters: the angles defining the camera axis orientation. Methods like SIFT and SURF normalize the translation and rotation component and simulate the scale (zoom) through image pyramids to obtain a description invariant to these parameters and partially invariant to affine transformations. *A-SIFT* (for *Affine SIFT*) [Morel and Yu, 2009] attempts to obtain a description fully invariant to affine transformations. The method simulates all image views obtainable by varying the

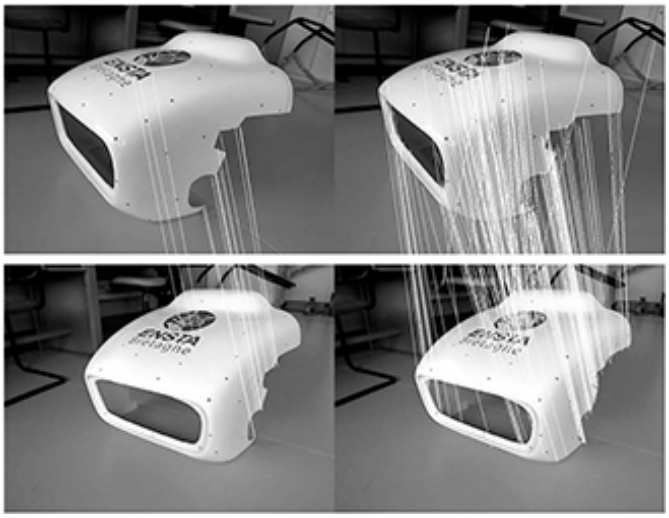

Figure A.3: Comparison of the number of correspondances found on an ENSTA mechanical prototype between SIFT on the left and A-SIFT on the right.

latitude and longitude camera angles. If a physical object has a smooth or piecewise smooth boundary, its images obtained by cameras in varying positions undergo smooth apparent deformations. These deformations are locally well approximated by affine transforms of the image plane. In consequence the solid object recognition problem has often been led back to the computation of affine invariant image local features. Such invariant features could be obtained by normalization methods, but no fully affine normalization method existed before. Yet the similarity invariance (invariance to translation, rotation, and zoom) is dealt with rigorously by the SIFT method. By simulating on both images zooms out and by normalizing translation and rotation, the SIFT method succeeds in being fully invariant to four out of the six parameters of an affine transform.

ASIFT is therefore much more efficient for our purposes. Moreover we believed it was possible to retrieve the rotation and translation parameters computed by ASIFT to obtain an estimation of those parameters to use them in our algorithm in the next chapter. However we haven't being able to do so yet.

A.3.2 System of equations

In the next part of this appendix, we solve the equations of the transformation between two poses using interval analysis and constraints propagation [Jaulin et al., 2001a]. Let's consider the following definition of the transformation matrix:

$$T = \begin{pmatrix} R & t \end{pmatrix} \tag{A.5}$$

where:

T is the transformation matrix,

R is the rotation matrix,

t is the translation vector.

The transformation T is estimated such that for each couple of corresponding points i and j, ideally:

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} - T \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} = 0 \tag{A.6}$$

where Xi, Yi and Zi are the coordinates of a point at the first pose and Xj, Yj and Zj are the coordinates of the corresponding point at the second pose. We define the translation vector as

$$t = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T \tag{A.7}$$

and the rotation matrix as the standard orthogonal matrix corresponding to a clockwise/left-handed rotation with Euler angles ϕ, θ, ψ with $x - y - z$ convention:

$$R = \begin{pmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{pmatrix} \tag{A.8}$$

Developing A.8 gives us three equations (or constraints) for each corresponding points.

$$\begin{aligned} (C_1) : X_i - (\cos\theta\cos\psi.X_j + (-\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi).Y_j \\ + (\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi).Z_j + t_x) = 0 \end{aligned} \tag{A.9}$$

$$\begin{aligned} (C_2) : Y_i - (\cos\theta\sin\psi.X_j + (\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi).Y_j \\ + (-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi).Z_j + t_y) = 0 \end{aligned} \tag{A.10}$$

$$\begin{aligned} (C_3) : Z_i - (-\sin\theta.X_j + (\sin\phi\cos\theta).Y_j + (\cos\phi\cos\theta).Z_j \\ + t_z) = 0 \end{aligned} \tag{A.11}$$

A.3.3 Forward-backward Algorithm

Let us remind that in our problem of finding the transformation parameters, we have three constraints per couple of corresponding points. They have to be treated simultaneously in a forward-backward algorithm to find the smallest boxes $[\overline{x}] = [\overline{\phi}].[\overline{\theta}].[\overline{\psi}].[\overline{t_x}].[\overline{t_y}].[\overline{t_z}]$ which encloses the solution set. We applied a forward-backward algorithm as presented in Chapter 2. Due to its size of 116 lines we haven't enclosed the full algorithm in this chapter. It can however be found at http://aymericbethencourt.com.

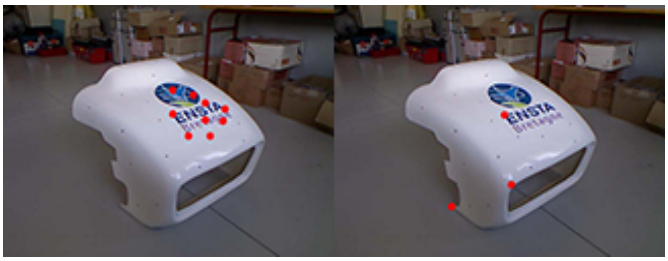

Figure A.4: Selection of (a) 10 almost collinear correspondances, and (b) 3 non-collinear correspondances

A.3.4 Result

We implemented our algorithm in C++ using *Luc Jaulin*'s interval library. The main advantage of an interval approach is generally its speed for solving strongly non-linear systems of equations as long as we have more equations than unknown variables. In our problem, we have six unknowns and three equations per corresponding points. This means that we only need two couples of corresponding points to solve the problem using interval analysis. In practice, all correspondence found by A-SIFT are sent to the forward-backward algorithm in order to maximize contractions. We first obtained what seemed to be random results, as we were feeding the algorithm with as many couples of points as it needed to contract the intervals to a acceptable width of $0.1rad$ on the rotation angles and $0.05m$ on the translation parameters. To reach this goal, our algorithm sometimes needed 200 couples of points (which was a problem when we had less correspondences found by A-SIFT) and sometimes needed as little as 3 points to attain this precision, making the computation time varying from 0.13 to $6.1ms$ ($0.1ms$ to initialize and $0.03ms$ to compute the forward-backward algorithm per corresponding couple of points). We eventually figured out that it depended on where the correspondences where located on the 3D model.

Fig. A.4a shows 10 almost collinear correspondences with which our contractors were not contracting well. We needed 200 of these points to get exploitable results. However, when using as little as 3 points that were clearly not collinear (fig. A.4b) we immediately contracted the intervals to the requested width. This was later explained as every isometry is completely determined by its effect on three independent (not collinear) points.

Fig. A.5 shows two PNG pictures and point clouds taken from a "right pose" and a "left pose" around the mechanical prototype. For this we used the *Robot Operating System* (*ROS*) with the *OpenNI* drivers.

We ran the PNG pictures of the two poses in ASIFT to obtain the correspondences. The algorithm computed for $45s$ on a Intel core 2 duo and found 287 correspondences over the structure.To compare the results, we also ran SIFT which took only $11s$ but found only 13 correspondences. We also noticed that they were all on the hood of the prototype meaning that the points were almost collinear and that our forward-backward algorithm would have failed. SIFT therefore requires to take intermediary poses to reconstruct the structure, which finally isn't making the use of SIFT faster than A-SIFT. However, existing reconstruction programs like *RGBDSLAM* use a parallel version of SIFT called *SIFT GPU* using the *Nvidia's CUDA* technology which considerably reduce its computation time. We decided to keep only 3 correspondences that we judged good enough (red points in fig.A.5). We eventually implemented a way for the program to automatically keep 3 points that were clearly not collinear: Among the correspondence points, we randomly

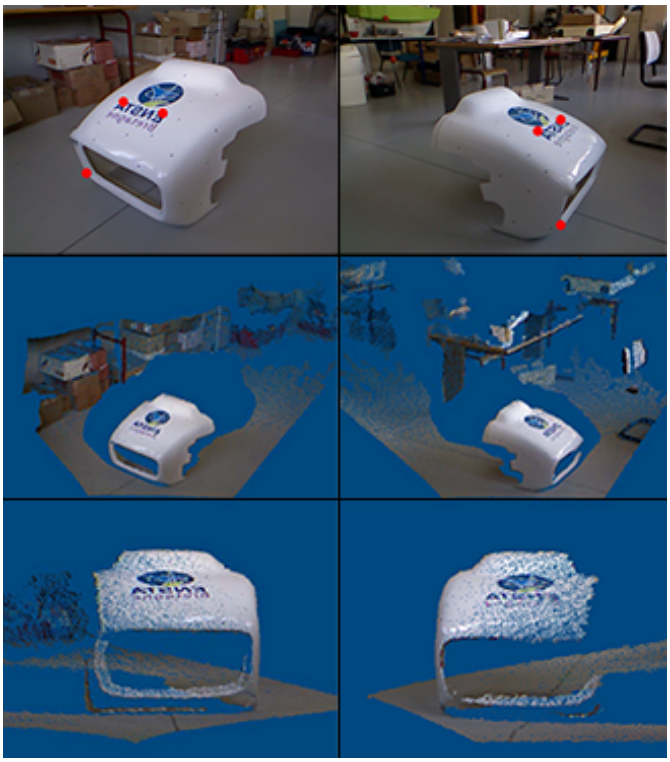

Figure A.5: *Kinect* capture of two poses.

choose 3 points. If they are almost collinear, we discard one point and randomly pick another one, and check again if they are almost collinear until they are not. Points can be shown to be almost collinear by determining that the scalar product of two vectors formed by the points is close to 0. In practice, we chose a arbitrary threshold depending on the performances we wanted to achieve. Once we had 3 clearly not collinear corresponding points, we recovered the depth information in the point clouds, converted the points into world coordinates and ran the points one after the other through our forward-backward algorithm, successively contracting $[\phi], [\theta], [\psi]$ and $[t_x], [t_y], [t_z]$ to the requested width. We first started with big intervals, for instance, $[\psi] = [-3.14, 3.14]$ and $[t_x] = [-10, 10]$. The program then outputted the contracted intervals after each pass of the forward-backward algorithm:

Point 1 $[\psi] = [-3.122, 2.593], [t_x] = [1.241, 1.489]$,

Point 2 $[\psi] = [-1.523, -1.623], [t_x] = [1.322, 1.412]$,

Point 3 $[\psi] = [-1.572, -1.573], [t_x] = [1.345, 1.385]$.

The intervals are repeatedly contracting around the solution and reached the requested width in $0.209ms$. Fig.A.6 shows the reconstructed scene using the two point clouds set with the computed transformation. We applied the same principle to 7 different poses taken around a car and displayed the result in fig. A.7.

Notice that the method can be made robust to outliers using the q-intersection presented in Chapter 2.

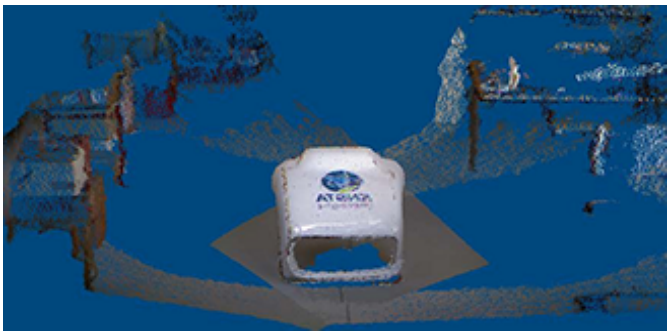

Figure A.6: The reconstructed structure from the two poses using the computed transform.

Figure A.7: My own car reconstructed from 7 *Kinect* poses. A video of the capturing process is available on http://youtu.be/GZHYMGErA6E and the reconstructed model on http://youtu.be/HKuSv8X3UWM.

Figure A.8: The IMU is mounted on the *Kinect.*

A.4 Adding an IMU

A.4.1 Why ?

The *IMU* or *Inertial Measurement Unit* contains an accelerometer, a magnetometer and a gyroscope. These last two combined allows us to obtain the orientation of the IMU (and thus the Kinect) at any time, therefore providing us with the rotation parameters between two poses (or at least a small interval around it). We used the *IMU-UM6* from *CHRobotics* and fixed it to the *Kinect* as shown in fig. A.8. In practice, the IMU turned out to be very precise about its orientation. According to the datasheets, this model was precise to $+/-0.035rad$, which was more than sufficient for our purposes. With this accuracy, contracting the intervals on ϕ, θ, ψ was not needed anymore. However, an IMU doesn't return its position so we still had to apply our forward-backward algorithm to compute the translation parameters. The IMU still improved our performances since only one correspondence was then needed to contract these parameters to an acceptable precision of $0.05m$. The computation then dropped to $0.1ms$to initialize plus $0.03ms$ to run the forward-backward algorithm once. Moreover the need to use only one correspondence made it possible to use less effective but faster algorithm than A-SIFT, like SIFT. Further studies could even lead to the implementation of a very fast algorithm that would only compute one correspondence and stop.

A.4.2 Position from acceleration

In order to discard all computation about solving the transformation parameters, we tried to obtain the position from the acceleration data from the IMU and reconstruct the 3D scene according to them only. By integrating twice the acceleration, we could theoretically find the translation between two poses. Knowing that the *Kinect* (thus the IMU) was at zero speed at $t = 0s$ allowed us to get rid of the constants. We also decided to stop the movement at each poses, meaning that the *Kinect* would also be at zero speed at the end of the movement. We imagined an algorithm based on the forward-backward principle: While moving, the estimated intervals of the speed of the *Kinect* are growing (see fig. A.9a), which, when integrated lead to a relatively imprecise estimation of the position with an important drift. Using the fact that the Kinect was at zero speed at the end of the movement, we contracted the speed intervals backward and obtained a better estimation of the position (see fig. A.9b).

As interval analysis is a "guaranteed" method, if, at line 6, $\{0\}$ is not included in the speed interval at the end of the movement, it means that the IMU is not stopped. In practice, the noise on the measures was very important because of the acceleration from gravity. If the Kinect stayed horizontally (like moving on a

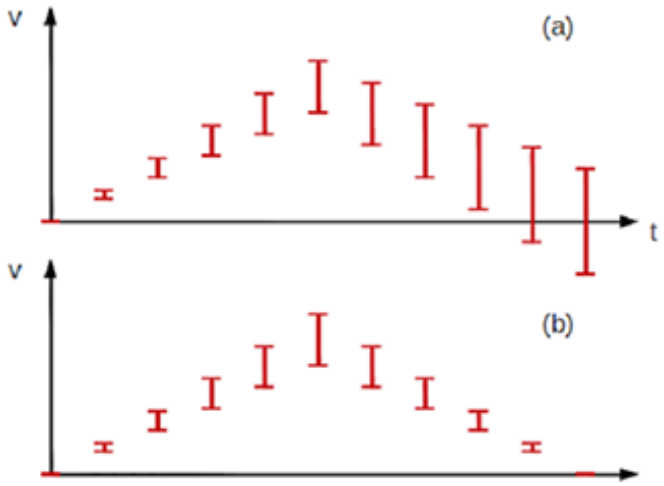

Figure A.9: Representation of the speed intervals (a) without zero speed at the end of the movement (b) with zero speed at the end of the movement..

	Algorithm $C_{ACCEL2POS}$ (in : $[a]$, out : $[x]$)
1	$[a]^0 = \{0\}; [v]^0 = \{0\}; [x]^0 = \{x_0\}; k = 0;$
2	for $t = 0 : dt : T - dt$
3	read $[a]^t$;
4	$[v]^{t+dt} = [v]^t + [a]^t * dt;$
5	endfor
6	if $\{0\}$ is not included in $[v]^T$ then error;
7	else
8	$[v]^T = \{0\};$
9	for $t = T : -dt : dt$
10	$[v]^{t+dt} = [v]^{t-dt} \backslash ([v]^t - [a]^t * dt);$
11	enfor
12	for $t = 0 : dt : T - dt$
13	$[x]^{t+dt} = [x]^t + [v]^t * dt$
14	endfor
15	endfi

Table A.1: Forward-Backward Algorithm for computing pose from acceleration

table), we could subtract the acceleration from gravity from the input acceleration on the z axis. However, in any other cases, we had to project the gravity on the estimated axis of the IMU. Although small, the noise and inaccuracy on theses axis was amplified by the double integrations which made the position very inaccurate. Translating the IMU of $1m$ without turning it already generated an error of $+/-11cm$. If we rotated the *Kinect* at the same time, the error jumped to $+/-70cm$, making the measure completely unexploitable.

A.5 Conclusion

Our research showed that it was possible to apply interval analysis to find the transformation between two 3D images. We have been able to reconstruct object and scenes in 3D. The addition of the IMU showed that it was possible to do so without any computation. However, we still have to figure out how to robustly compute the position of the IMU at any time. Further studies may include loop closure detection [Aubry et al., 2013] and the combination with probabilistic methods [Lemaire et al., 2005]. Notice that we tried to integrate the reconstructed car into a game engine. However, we had to convert the points cloud into a mesh. We tried different algorithm in different software but the results were either containing too many polygons for a game engine to run it, or too simplified to still look like a car. This problematic is a very active topic of research in the graphics world. A company named *JCL* found a side solution by developing a game engine exclusively for displaying point clouds of trillions of points in real-time. It would therefore be easy for developers to scan an object or a scene with the *Kinect* (or any 3D scanner) and include it in a game. Finally, we believe that it is possible to build complete robot navigation and interaction systems solely based on cheap depth cameras like the *Kinect*, especially since vison-based SLAM has already been achieved using panoramic images [Lemaire and Lacroix, 2007] and stereovision images [Lemaire et al., 2007]. Further work will include mounting a *Kinect* with an IMU on an autonomous robot to perform 3D SLAM.

The work presented in this chapter has been **published** in : *InTech, International Journal of Advanced Robotics*, 2012.

Appendix B

Design and experimental validation of a visual goniometric localization system for a group of indoor robot vehicles

As the final project of my engineering degree, I designed a visual goniometric localization system for the CAROTTE competition. As the subject is relevant to this thesis, this appendix presents my work.

B.1 Introduction

The *CAROTTE* (*CArtographie par RObot d'un TErritoire*) competition has been created by the *DGA* (*Direction Général pour l'Armement*) for military purposes. The goal is for a group of robots to explore an unknown building and perform SLAM while detecting various pre-defined objects. In order to take part in the competition, we built robots designed to detect and localize each other in the arena. Each robot have an rotating laser that enable it to map and localize himself in his visible environment. To strengthen the localization, robots can put their maps in common to aggregate them into a global map. However to build the aggregated map, each robot's map must be put in relation with each other. For this, each robot must know the distance, angle and azimuth to other robots. Using camera calibration, we developed a new visual goniometric localization system that put a robot in geometrical relationship with another when its camera sees it.

B.2 Realization

When a robot sees another, it must be able to compute the distance and azimuth to the robot, and the angle of the observed robot. For this, a simple system of multiple *LEDs* (*Light-Emitting Diodes*) on each robot is sufficient. Using methods of camera calibration, the distance on screen between each observed LED can lead to the real geometric parameters in the real world.

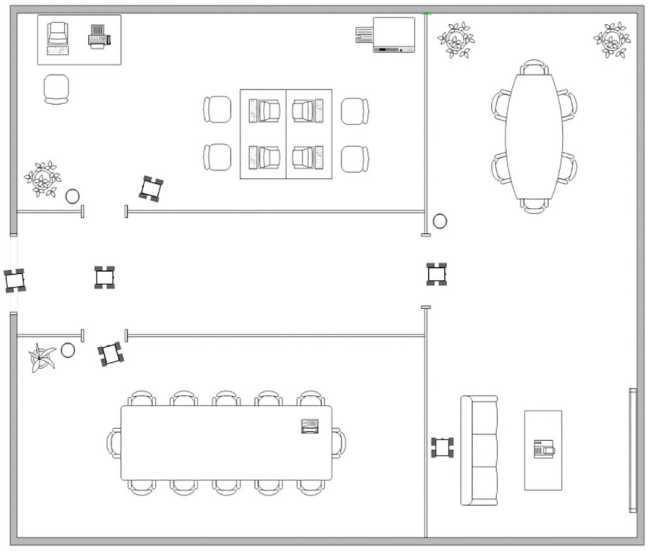

Figure B.1: A group of robot moving in the arena.

B.2.1 System design

In order to realize the desired function, we first imagined two hoops over the robot in order to form a tetrahedral structure with 4 LEDs. The three LEDs at the base would allow to compute the angle of the observed robot, and the LED at the top (which could be on the same color of a base LED) would allow to compute the distance to the robot by measuring the distance with the base LEDs. It was important to always see at least the top LED and two base LEDs.

We first designed several early views of the project, e.g. Fig. B.2, which showed multiple flaws, like the impossibility to open the on-board computer anymore. We therefore considered an concept of LED tower as presented in fig. B.3(a) with the 4 LEDs mounted height on a tower. In order to maximize the segmentation of the LEDs' light, we used 200 degrees ultra-bright LEDs.

However the problem with an ultra-bright LED is that the center is so bright that it saturates the *CCD* sensor of a webcam. The result on picture was a white center with a diffuse colored halo, making it hard to segment because of its intricate and skewed form in certain directions. Segmenting fig. B.3(a) was disastrous. We then had the idea to use a concept inspired from the *Playstation Move* presented in fig. B.4. The *Move* is a game controller used with a webcam placed on top of the TV. The webcam tracks the movements of the player thanks to an opaque sphere, which light is clear and uniform.

Using ping-pong balls cut in half, we were able to create a second version of the LED tower shown in fig. B.3(b). The semi-transparent matter of the balls acted as a filter to the ultra-bright light, giving the luminous halos a spheric shape and uniform color just like the *Playstation Move*. Computing the barycenter of these luminous areas would therefore be much more precise.

The first segmentation tests of this LED tower rapidly showed a new issue. The LEDs were way too close from each others. With a webcam resolution of $640 \times 480px$, the barycenters of two LEDs were only 3 pixels away on screen when the robot was $5m$ away from the webcam. This would have been disastrous for the

Figure B.2: Early project view.

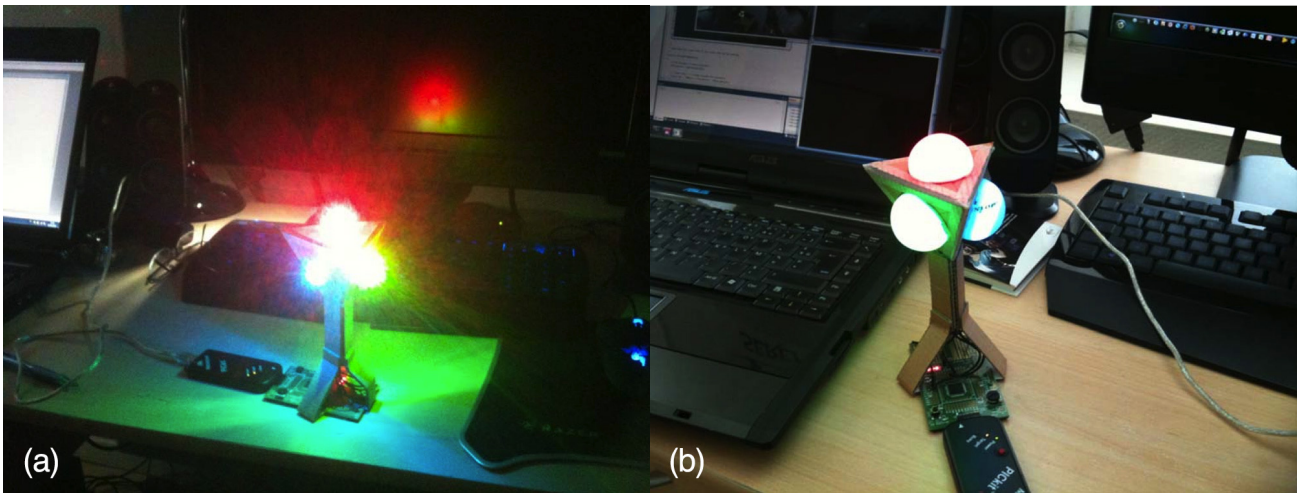

Figure B.3: (a) First version of the LED Tower. (b) Second version of the LED Tower.

Figure B.4: The Playstation Move (© Sony Entertainment)

precision of the measures during the competition; so we increased the space between the LEDs. We modified the concept of the robot and added the LEDs directly on it. Using PVC tubes, we were able to place the LEDs as far away from each other on the robot chassis and built the first prototype in fig. B.5(a).

This version kept the tetrahedral shape of the LED tower but with the robot chassis and depending on the point of view, it was not always possible to see three LEDs at the same time. By viewing only two LEDs, it would be impossible to compute either the distance or the angle of the robot. After multiple tryouts, we found a way an optimize the LEDs arrangement. We even removed one of the LED at the bottom and moved the other two like presented in fig. B.5(b) and (c). This way, the three LEDs were almost always visible regardless of the angle of the robot.

The difference in height (on the y axis) between the LED on top and the two lower LEDs allowed to compute the distance to the robot, and the arrangement of all three LEDs on the x axis allowed to compute the angle of the robot. Finally, the barycenter of all LEDs would allow to computer the azimuth of the robot.

We first believed that there would be an issue when one of the tube would hide a LED, but as the blue LED is higher, only two angles of the robot can hide the green or red LED, in which case we actually perfectly know which angle engenders the occlusion.

A few tests on this prototype were very conclusive, and so the "assembly line" began. Seven platforms were built.

B.2.2 Coding the software

OpenCV Implementation

Each robot embedded a *Core 2 duo eee-pc* to run the localization program, and a *PIC 16F887* to make the LEDs blink at $4Hz$. We coded the localization program in C using the *OpenCV* library. The program took an image every $250ms$ to get a picture with LEDs off (fig. B.7(a)) and LEDs on (fig. B.7(b)). The program then took the difference between both images in order to keep only the LEDs (fig. B.7(c)). The image is then segmented to identify the colored clusters. In order to work for all light intensity, we first decided to work in *HSV* (*Hue Saturation Value*). We took HSV measurements from each LED as seen by the webcam and tried computing 95% enclosing spheres. Unfortunately, the measures proved to be very entangled and hardly separable. We then remembered that we first chose the HSV model to easily segment a yellow LED used in previous prototypes. We then made new measurements in *RGB* (*Red Green Blue*) presented in fig. B.6.

The values were more separated and segmenting the colors were easier. However depending on the ambient luminosity, the RGB values change a lot in a non-linear way (not the case in HSV). These variations were even more present when the webcam was adjusting it contrast and gain automatically like most webcam. We had to modify the pilots of the webcam to de-activate this functionality. In the meantime, we took the opportunity to adjust the webcam gain settings to accentuate bright colors on each red, green or blue channel and make the segmentation even more powerful. Fig.B.7(d) shows the segmentation result. From this picture, it was easy to compute the barycenter of each LED (fig.B.7(e)).

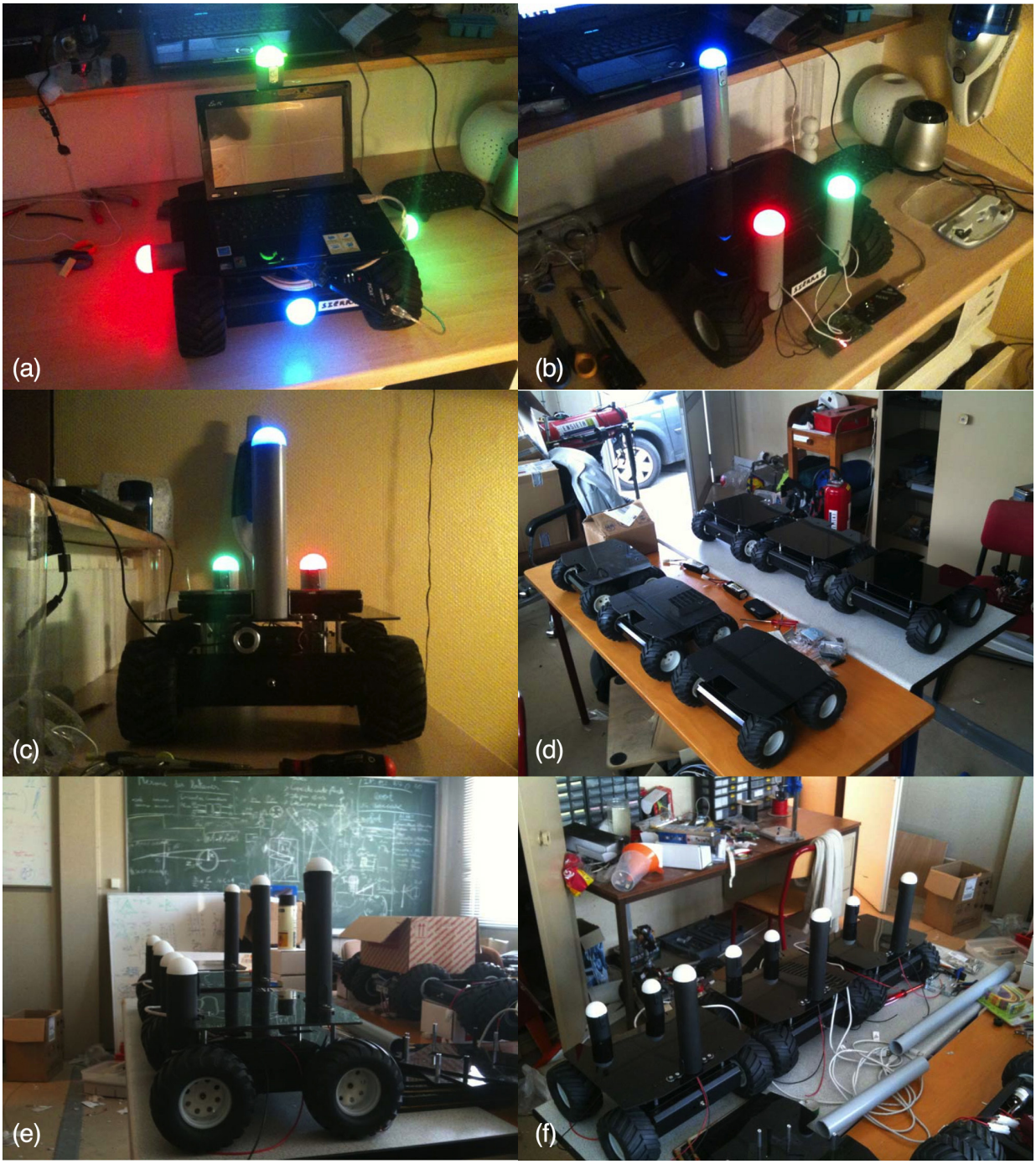

Figure B.5: (a) First prototype. (b) Second prototype. (c) Second prototype as seen by another robot. (d) The assembly line. (e) Three completed robots. (f) The robots were built identical.

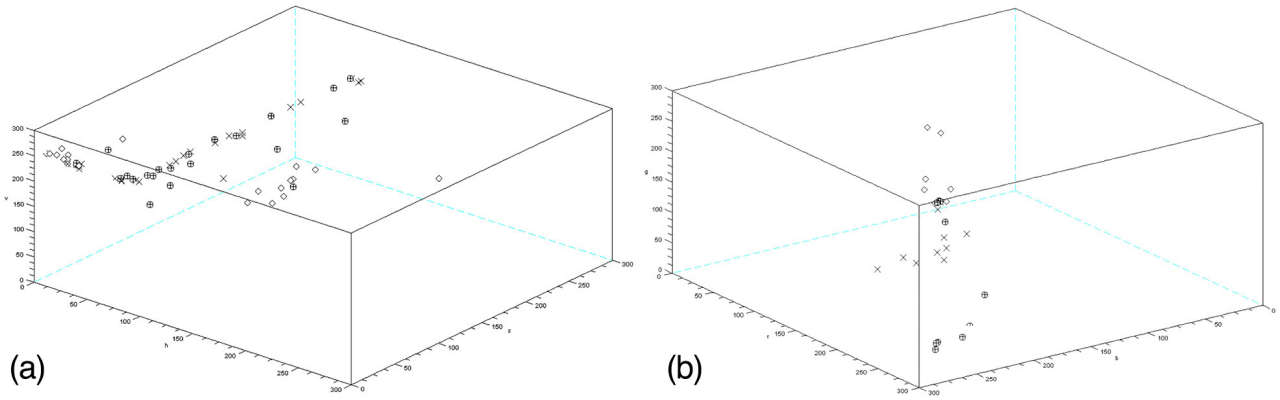

Figure B.6: (a) HSV measurements. (b) RGB measurements. The lozenges represents the red LED, the crosses the blue LED and the circled crosses the green LED.

It was therefore possible to compute the distance, angle and azimuth of the observed robot.

Distance measurement

The distance was simply computed by applying :

$$hpix/h\,\text{real} = \delta/d\,\text{real} \tag{B.1}$$

where :

$hpix$ is the distance on screen in pixels between the top blue LED and any of the lower LEDs,

h real is the real distance,

$dpix$ is the distance from the robot to the webcam (that we look to compute),

δ is a calibration parameter that needed to be calibrated.

Therefore $d\,\text{real} = \delta * h\,\text{real}\,/hpix$. We found the calibration parameter δ experimentally. For this, we realized the setup presented in fig. B.8 and realized measurements presented in table B.2.

$hpix$ (pixels)	$dreel$ (cm)
63	100
45	150
34	200
29	250
24	300
19	350
18	400
16	450

(B.2)

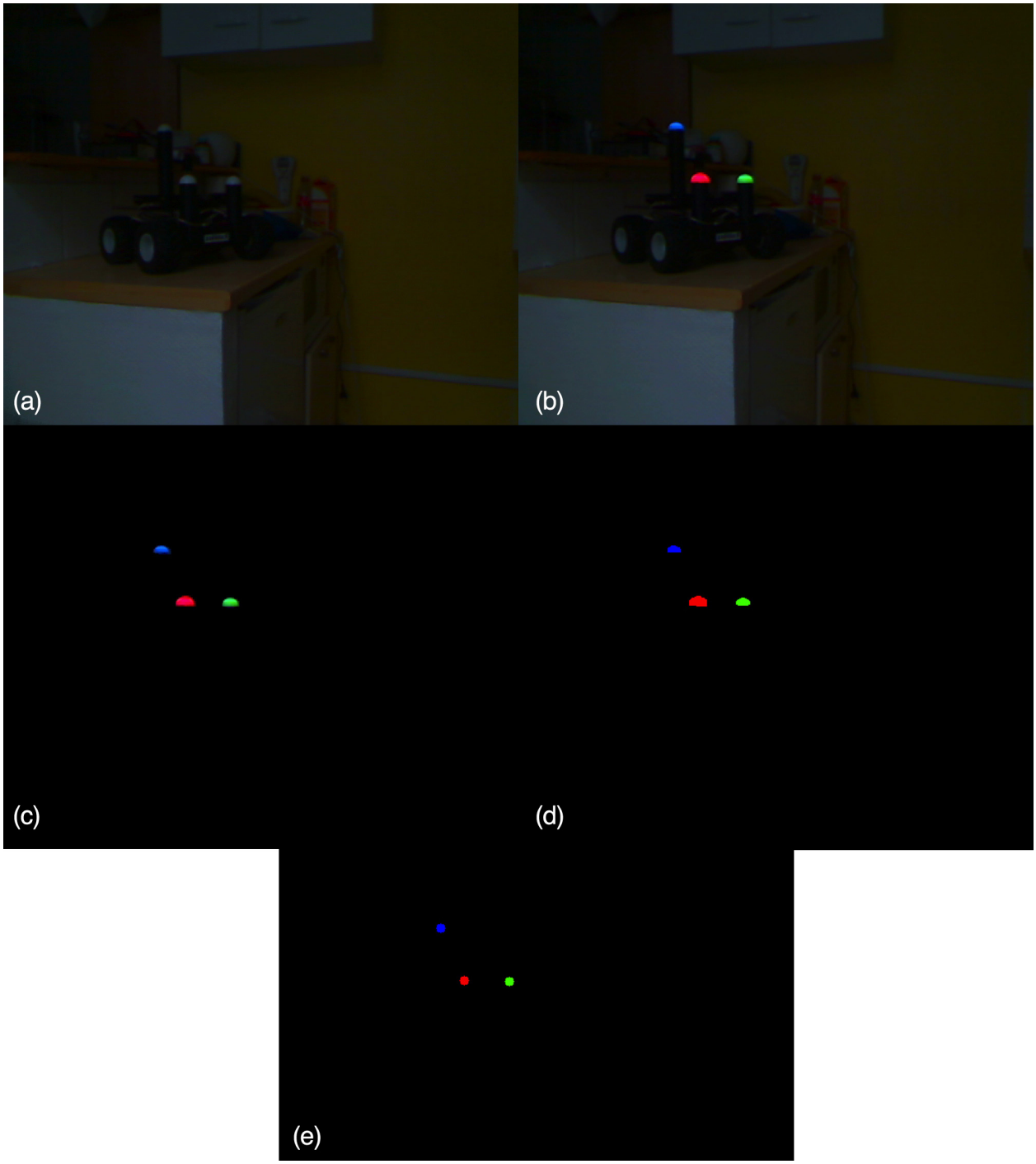

Figure B.7: (a) At $t = 0ms$, we get a picture with the lights off. (b) At $t = 250ms$, we get a picture with the lights on. (c) Subtration of (a) to (b) .(d) Segmentation of the colors. (e) Barycenters of each segmented areas.

Figure B.8: Simple representation of the distance measurement.

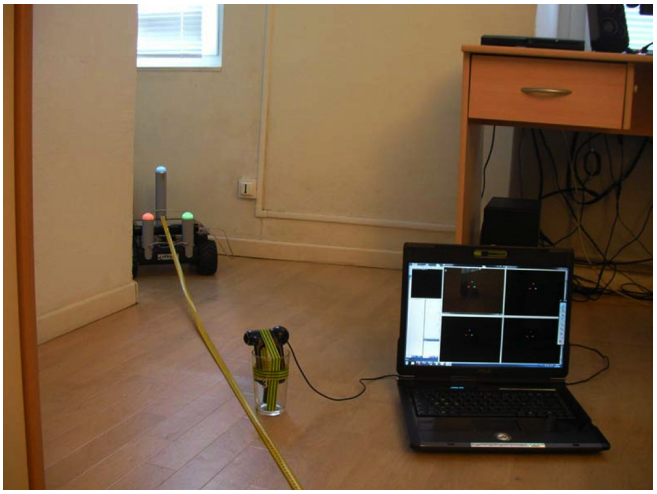

Figure B.9: Calibration between the real distance to the robot and observed difference of height between the LEDs on screen.

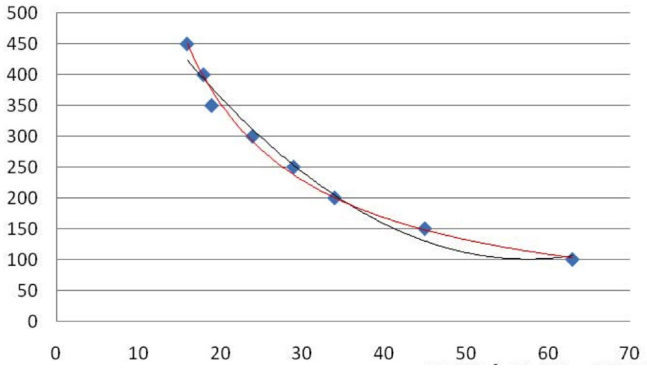

Figure B.10: Polynomial interpolation of the data.

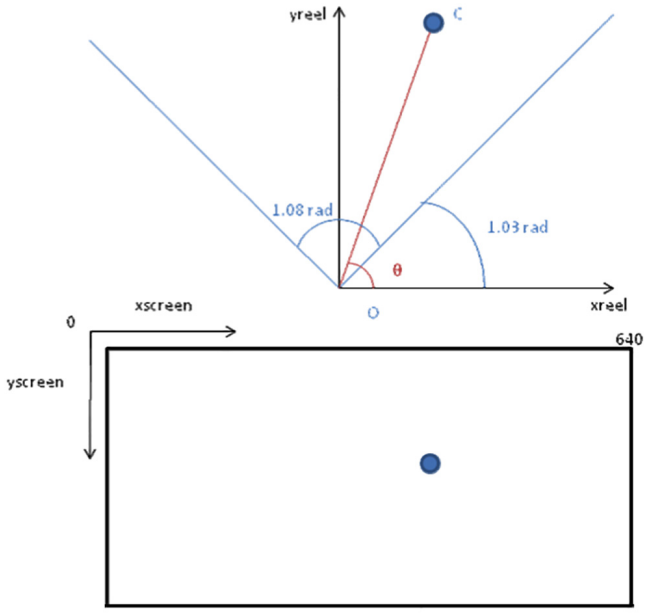

Figure B.11: Geometrical representation of the azimut problem. The upper image represents the real world, and the lower image the image observed by the webcam.

Fig.B.10 presents the polynomial interpolation that was later done on the data, to compute the model that best fitted the data. We deduced that $\delta = 886.74$. After implementing this equation into the program, a few tests showed that it was possible to compute the distance to the robot with a precision of $+/-4cm$.

Azimuth measurement

We then had to find the equation to compute the azimuth θ of the robot. The webcam we were using had an aperture angle of 62 degree or $1.08rad$ according to its datasheets. The image on screen was 640 pixels large. As the position on screen of a point C moves linearly according to its angle to the webcam in the real world (fig. B.11), we could easily deduce that:

$$\theta = (640 - x) * 1.08/640 + 1.03 \tag{B.3}$$

where θ is the azimuth of the robot C in relation to the webcam centered on its ordinate axis and x the abscissa on screen of C in pixels.

During the tests presented in fig. B.12, this equation proved to be precise to $+/-0.07rad$. Knowing the azimuth and distance of the robot, it was already possible to map the position of every robot seeing each others.

Orientation measurement

During the tests above on different azimuths, we also turned the robot to different orientations δ and took multiple distance measurements on screen between the LEDs. The result for an azimuth of 90 degrees and

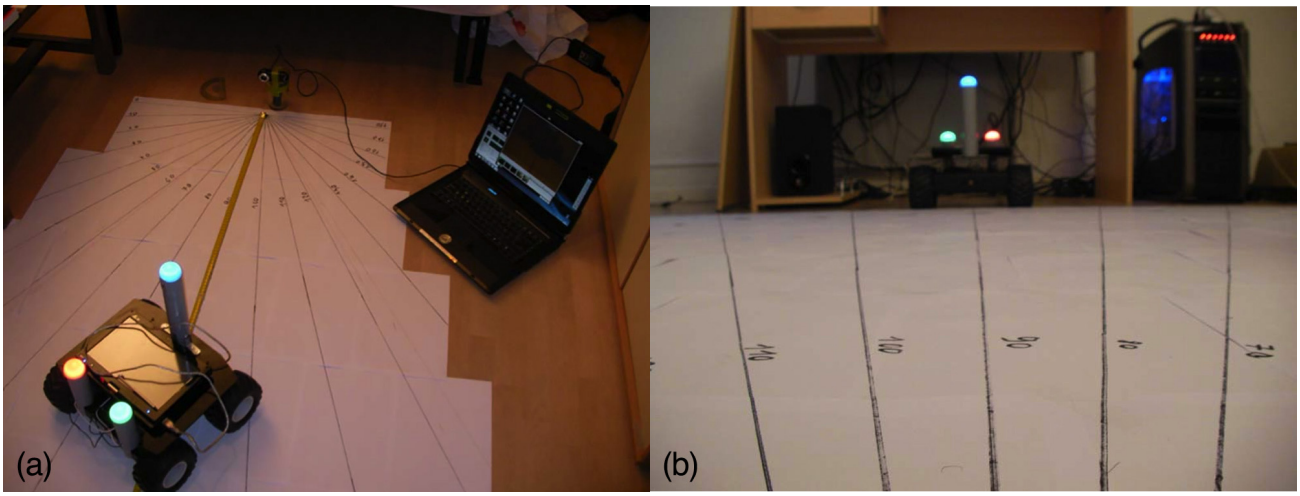

Figure B.12: (a) Azimut calibration tests using a giant handmade protractor. (b) View from the webcam.

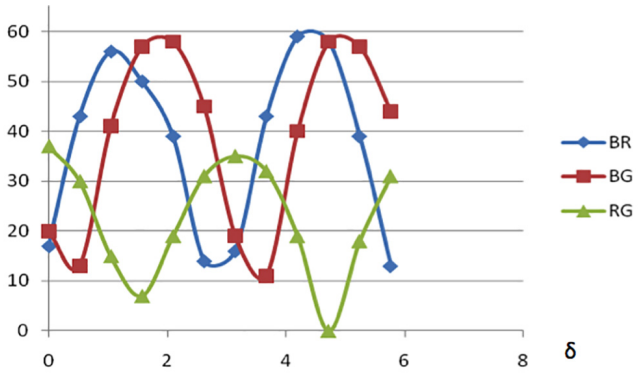

Figure B.13: Distances in pixels between the blue (B), green (G) and red (R) LEDs when turning the robot.

distance of $1m$ is presented fig.B.13.

Six orientation areas can be identified as presented in fig.B.14 for which we can assume that the orientation of the robot moved in a linear fashion compared to the distance between the LEDs on screen.

Fig. B.15 presents for example the R-G-B area (the green LED moves between the red and blue LEDs). When the green LED is at maximum to the right side (the blue and green LED are aligned), we measured $\theta = 0.30rad$. When the green LED is at maximum on the left side (the red and green LED are aligned), we measured $\theta = 1.57rad$. We then simply computed a linear interpolation $\theta = BG/BR * (1.57 - 0.30) + 0.30$ where BG is the distance observed on screen between the blue and green LEDs, and BR is the distance between the blue and red LEDs.

This linear model was very approximated but still provided good results during tests with a precision of $+/- 0.11rad$. We then simply coded an interface (fig. B.16) to localize the robot on screen.

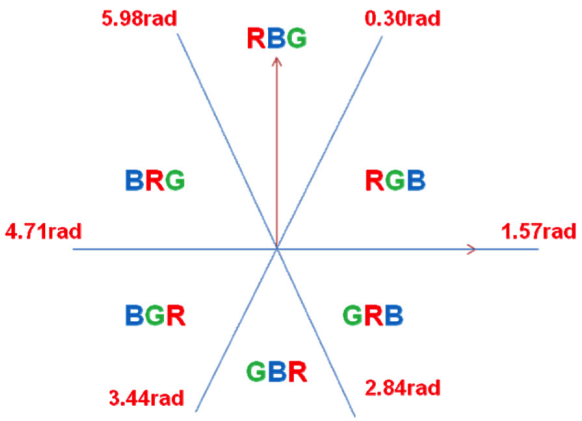

Figure B.14: The 6 orientation areas of the observed robot in reference to the observer.

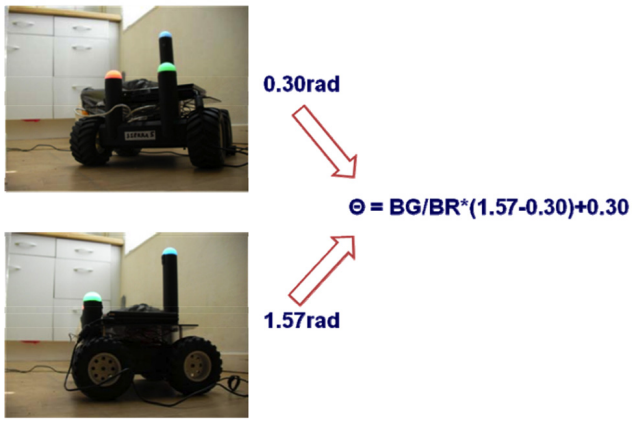

Figure B.15: Example of the computation of the linearized equation of the orientation of the robot, here in the R-G-B area.

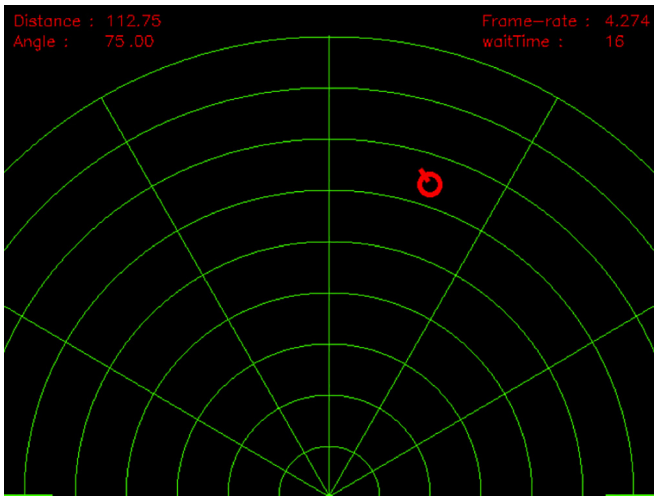

Figure B.16: The HMI of the localization program.

B.3 Conclusion

In this appendix, we showed the work realized for the robotic competition CAROTTE. We developed a new goniometric method to visually put robots in geometrical relationship with each other. Using 3 LEDs mounted on the robots, we used camera calibration techniques to compute the distance, orientation and azimuth of the observed robot. Further developments will include the segmentation of multiple robots observed at the same time on screen and the grouping of each 3-upplets of LEDs. Finally, notice that this approach only provides a relative localization of the robots with each others, and could be combined with an odometric system [Bonnifait and Garcia, 1998] for a global localization.

The work presented in this chapter has been **submitted** to : *Revue de l'électronique et de l'électricité*, 2014, and is currently awaiting reviews.

Appendix C

Résumé en français

C.1 Introduction

La localisation est un sens essentiel à toute créature terrestre visant à se déplacer dans son environnement. Les organes ou senseurs utilisés par les humains sont en partie les yeux et le système vestibulaire (oreille interne). Tout comme les humains, les robots ont besoin de capteurs pour se localiser, respectivement une caméra et un gyroscope. Cependant les machines n'ont pas encore les capacités du cerveau humain pour se localiser. C'est pourquoi il est important de développer des algorithmes efficaces de localisation, et plus particulièrement sous l'eau où il n'est pas possible d'utiliser le GPS. Les *AUVs (Véhicules sous-marins autonomes)* sont de plus en plus développés dans les domaines civils (recherche sismique, inspection d'installations marines, etc.) et militaires (guerre des mines). Je fus personnellement interviewé par France 2 sur ce sujet qui s'étend de manière exponentielle (interview disponible sur http://youtu.be/Zwjbufay9Z0). L'avantage des AUVs comparés aux traditionnels *ROVs (Véhicules télé-opérés)* est leur habilité à se "débrouiller tout seul". La mission est chargée dans l'AUV et la réalise sans connexion à l'opérateur. Une bonne localisation de l'AUV est alors essentielle.

A cela s'ajoute des problématiques de monté en échelle. Certains projets développés par la *Direction Général pour l'Armement*, ou de grandes compagnies telles que *CGG* ou *Saudi Aramco* prévoient la mise en place d'essaims de plusieurs milliers d'AUVs simultanément afin de réaliser des mesures sismiques. Les AUVs doivent alors se positionner en grille de manière très précise. L'utilisation de méthodes dites par intervalle semble donc justifiée dans ce contexte où la localisation de l'AUV à besoin d'être garantie.

En effet, les méthodes par intervalle permettent de représenter des ensembles solution avec leur incertitudes, garantissant qu'aucune solution n'existe en dehors de ses bornes. Nous verrons également que l'analyse par intervalles possède de nombreux outils particulièrement éfficaces pour notre problématique. Les méthodes par intervalle sont notamment très puissantes pour la résolution de larges systèmes d'équations, particulièrement adaptés dans le cas de milliers d'AUVs.

Cette thèse est organisée comme suit. Le chapitre 1 introduit le sujet. Le chapitre 2 présente l'analyse par intervalle et son application à la localisation en robotique mobile. Le chapitre 3 (non-présenté dans ce résumé) étend la notion de contracteur et propose l'évaluation de contracteurs minimaux par transformations

géométriques. Le chapitre 4 formalise le problème de localisation dynamique d'AUVs sous la forme d'un problème de satisfaction de contraintes, et propose un nouvel outil, le tube, pour résoudre ce problème. Le chapitre 5 s'intéresse ensuite à un modèle d'AUV plus proche de la réalité, pour lequel l'horloge du robot est désynchronisée par rapport au temps réel. Enfin le chapitre 6 conclu la thèse par une forte monté en échelle des algorithmes et simulations présentés dans les chapitres précédent.

C.2 L'analyse par intervalle et son application à la localisation en robotique

C.2.1 L'approche ensembliste

Ce premier chapitre introduit les notions basiques de l'analyse par intervalle et démontre leur application sur un simple problème académique de localisation. Considérons le problème suivant : un robot de position incertaine peut mesurer sa distance à d'autres robots dont la position est connue. Le modelé d'observation est :

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{C.1}$$

où (x_i, y_i) sont les coordonnées du robot i, (x_j, y_j) sont les coordonnées du robot j et d_{ij} est la distance mesurée entre eux. On remarque que le problème est non-linéaire.

L'approche ensembliste (ou par intervalle) consiste à considérer que l'erreur de mesure est bornée et donc que la distance réel $d_{ij} \in d_{ij}^{measured} + [e_{ij}]$ ou $d_{ij}^{measured}$ est la distance mesurée et $[e_{ij}]$ est l'intervalle qui inclue l'erreur de mesure. La mesure de distance n'est alors pas représentée par un cercle autour du robot mais par un anneau. L'image C.1 illustre le principe avec quatre robots.

C.2.2 L'analyse par intervalle

Un *intervalle* $[x]$ est définie comme l'ensemble des nombres réels x entre une borne inférieure $\underline{x}$ et une borne supérieure $\bar{x}$.

$$[x] = [\underline{x}, \bar{x}] = \{x \in \mathbb{R}, \underline{x} \leq x \leq \bar{x}\} \tag{C.2}$$

Cette représentation permet de travailler avec des valeurs incertaines de manière fiable. Quand des contraintes (égalités ou inégalités) sur les variables sont disponibles, il est possible de contracter l'intervalle sans perdre de solution.

L'*intersection* de deux intervalles non vides $[x]$ et $[y]$ satisfait :

$$[x] \cap [y] = \begin{cases} [\max\{\underline{x}, \underline{y}\}, \min\{\bar{x}, \bar{y}\}] & \text{si } \max\{\underline{x}, \underline{y}\} \leq \min\{\bar{x}, \bar{y}\} \\ \emptyset & \text{sinon} \end{cases}$$

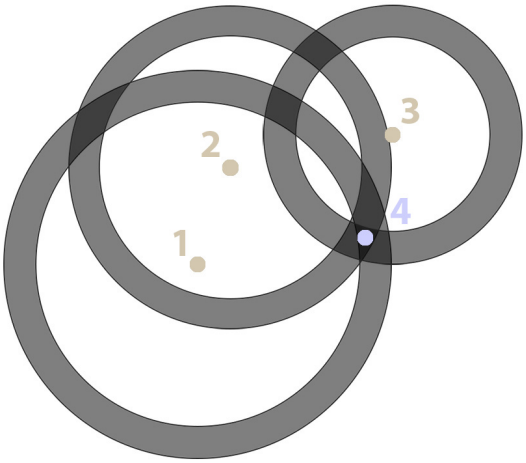

Figure C.1: Le robot 4 (en violet) se localise grâce à des mesures de distance à erreur bornée avec trois autres robots (1, 2 et 3 en marron).

Exemple : $[1,3] \cap [2,5] = [2,3]$

L'*union* de deux intervalles non vides $[x]$ et $[y]$ satisfait :

$$[x] \sqcup [y] = [\min\{\underline{x}, \underline{y}\}, \max\{\bar{x}, \bar{y}\}] \tag{C.3}$$

Exemple : $[1,3] \sqcup [5,7] = [1,7]$

Pour deux intervalles $[x]$ et $[y]$ et un opérateur $\diamond \in \{+, -, *, /\}$, nous définissons $[x] \diamond [y]$ comme étant le plus petit intervalle contentant toutes les valeurs possibles pour $x \diamond y$ telle que $x \in [x]$ et $y \in [y]$ où

$$[x] \diamond [y] = [\{x \diamond y \in \mathbb{R} | x \in [x], y \in [y]\}] \tag{C.4}$$

Pour des intervalles fermés :

$$[x] + [y] = [\underline{x} + \underline{y}, \bar{x} - \bar{y}] \tag{C.5}$$
$$[x] - [y] = [\underline{x} - \bar{y}, \bar{x} - \underline{y}] \tag{C.6}$$
$$[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}\}] \tag{C.7}$$

Exemple :

$$[-1,3]+[2,7]=[1,10] \tag{C.8}$$
$$[-1,3]-[2,7]=[-8,1] \tag{C.9}$$
$$[-1,3].[2,7]=[-7,21] \tag{C.10}$$

Une *boite* $[\mathbf{x}]$ de $\mathbb{R}^n$ est le produit cartésien de n intervalles.

$$[\mathbf{x}]=[x_1]\times ...\times [x_n]=[\underline{x}_1,\bar{\mathrm{x}}_1]\times ...\times [\underline{x}_n,\bar{\mathrm{x}}_n] \tag{C.11}$$

Les règles définies pour les intervalles valent aussi pour les boites.

L'image $f([x])$ d'un intervalle $[x]$ par une fonction f est

$$f([x])=\{f(x)|x\in[x]\} \tag{C.12}$$

L'image peut ne pas être un intervalle, par exemple si f n'est pas continue. L'extension intervalle est définie comme la fonction retournant l'enveloppe intervalle :

$$[f]([x])=[\{f(x)|x\in[x]\}] \tag{C.13}$$

L'extension intervalle des fonctions élémentaire peut être directement écrite par ses bornes. Par exemple, pour $[x]$ non vide, l'extension d'intervalle de la fonction exponentielle est :

$$[\exp]([x])=[\exp\underline{x},\exp\bar{x}] \tag{C.14}$$

De même, l'image d'une boite $[\mathbf{x}]$ par une fonction $\mathbf{f}$ n'est pas toujours une boite. Nous utiliserons alors la *fonction d'inclusion* qui inclue l'image de la boite. La fonction $[\mathbf{f}]:\mathbb{IR}^n\to\mathbb{IR}^m$ est une fonction d'inclusion pour $\mathbf{f}:\mathbb{R}^n\to\mathbb{R}^m$ si et seulement si

$$\forall[\mathbf{x}]\in\mathbb{IR}^n,\mathbf{f}([\mathbf{x}])\subset[\mathbf{f}]([\mathbf{x}]) \tag{C.15}$$

La fonction d'inclusion minimale $[\mathbf{f}]^*$ est definie comme la fonctions dont l'image est la plus boite incluant l'image de f. L'image C.2 illustre la notion.

Considérons maintenant n_x variables réelles $x_i\in\mathbb{R}, i\in\{1,...,n_f\}$ liées par n_f relations (ou *contraintes*) de la forme :

$$f_j(x_1,x_2,...,x_{n_x})=0, j\in\{1,...,n_f\} \tag{C.16}$$

ou f_j dénote la fonction pour chaque coordonné j. On sait que chaque variable x_i appartient à l'intervalle $[x_i]$. On definit $\mathbf{x}=(x_1,x_2,...,x_{n_x})^T$ et le domaine pour $\mathbf{x}$ comme $[\mathbf{x}]=[x_1]\times[x_2]\times...\times[x_{n_x}]$. On note aussi $\mathbf{f}$ la fonction dont les fonctions coordonnées sont f_j. On peut donc réécrire (C.16) sous sa forme vectorielle

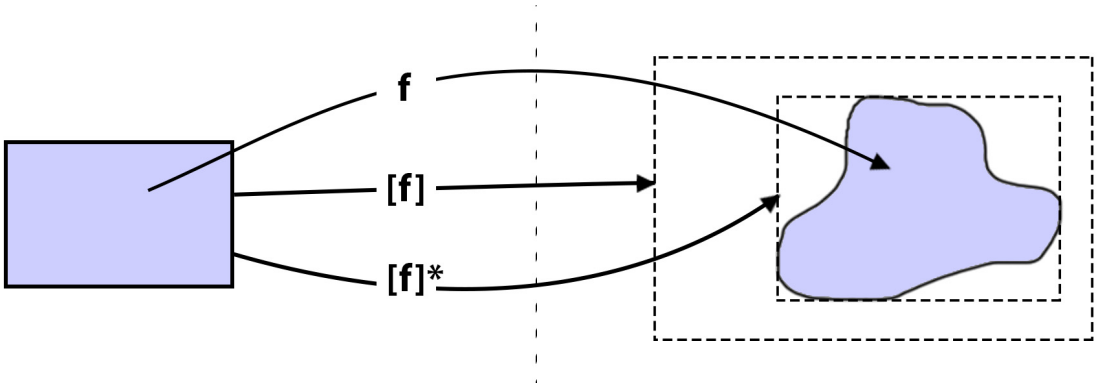

Figure C.2: Présentation de l'image d'une boite par la fonction $\mathbf{f}$, de la fonction d'inclusion $[\mathbf{f}]$ et de la fonction d'inclusion minimale $[\mathbf{f}]^*$.

$\mathbf{f}(\mathbf{x}) = \mathbf{0}$. On appelle alors cela un *problème de satisfaction de contraintes* (CSP en anglais) que l'on notera $\Im$.

$$\Im : (\mathbf{f}(\mathbf{x}) = 0, x \in [x]) \tag{C.17}$$

Un CSP est donc composé de variables, domaines contenants ces variables et de contraintes. La solution $\mathbb{S}$ de $\Im$ est définie comme suit :

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}] | \mathbf{f}(\mathbf{x}) = \mathbf{0}\} \tag{C.18}$$

Contracter un CSP, c'est remplacer le domaine $[\mathbf{x}]$ par un plus petit domaine $[\mathbf{x}']$ sans changer la solution. Notons que les domaines considérés dans cette thèse sont des intervalles de $\mathbb{R}$. Nous avons donc $\mathbb{S} \subset [\mathbf{x}'] \subset [\mathbf{x}]$. L'operateur permettant de contracter $\Im$ est appelé un *contracteur*. On définie le *contracteur minimal* comme le contracteur remplaçant $[x]$ par la plus petite boite contenant $\mathbb{S}$.

De nombreux problèmes en estimation, control, robotique, etc. peuvent etre représentés par des CSP [Araya et al., 2008],[Ceberio and Granvilliers, 2001] et de nombreux contracteurs minimaux sont défnies pour resoudre optimalement certaines classes de problème [Chabert and Jaulin, 2009] [Jaulin et al., 2001a]. Le contracteur que nous allons utiliser pour notre problème de localisation est le *contracteur par propagation et retro-propagation* [Benhamou et al., 1999] qui permet de contracter les domaines d'un CSP $\Im : (\mathbf{f}(\mathbf{x}) = 0, x \in [x])$ en isolant chaque contrainte séparément. Pour rapel, chaque distance mesurée par notre robot s'écrit :

$$d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{C.19}$$

La première étape du contracteur par propagation et retro-propagation est donc de décomposer cette contrainte en *contraintes primaires* en introduisant de nouvelles variables.

$$
\begin{aligned}
i_1 &= -x_j \\
i_2 &= x_i + i_1 \\
i_3 &= i_2^2 \\
i_4 &= -y_j \\
i_5 &= y_i + i_4 \\
i_6 &= i_5^2 \\
i_7 &= i_3 + i_6 \\
d &= \sqrt{i_7}
\end{aligned}
\tag{C.20}
$$

Initialement, les intervalles associés au variables i_k sont $]-\infty;\infty[$. La méthode pour contracter $\Im$ est de contracter chaque contrainte primitive jusqu'au point fixe [Waltz, 1975]. Pour les contraintes binaires (deux variables) tel que la racine carrée, deux étapes de contractions sont nécessaires : la contraction par l'image directe de la fonction et la contraction par l'inverse de la fonction. Dans notre exemple, la contrainte $d = \sqrt{i_7}$ se réécrit sont les formes :

$$d = \sqrt{i_7} \tag{C.21}$$
$$i_7 = d^2 \tag{C.22}$$

et les contractions qui en découlent sont :

$$[d] = [d] \cap \sqrt{[i_7]} \tag{C.23}$$
$$[i_7] = [i_7] \cap [d^2] \tag{C.24}$$

Pour les contraintes ternaires (reliant trois variables), il y a trois formes possibles de réécriture de la contrainte. Considérons la contrainte $i_7 = i_3 + i_6$ avec par exemple des intervalles initiaux $[i_3] = [-\infty, 2], [i_6] = [-\infty, 3]$ et $[i_7] = [4, \infty]$. Nous pouvons facilement contracter ces intervalles sans supprimer de valeur compatible avec la contrainte :

$$
\begin{aligned}
i_7 = i_3 + i_6 \rightarrow i_7 &\in [4,\infty] \cap ([-\infty,2] + [-\infty,3]) \qquad \text{(C.25)} \\
&= [4,\infty] \cap [-\infty,5] = [4,5] \\
i_3 = i_7 - i_6 \rightarrow i_3 &\in [-\infty,2] \cap ([4,\infty] - [-\infty,3]) \qquad \text{(C.26)} \\
&= [-\infty,2] \cap [1,\infty] = [1,2] \\
i_6 = i_7 - i_3 \rightarrow z &\in [-\infty,3] \cap ([4,\infty] + [-\infty,2]) \qquad \text{(C.27)} \\
&= [-\infty,3] \cap [2,\infty] = [2,3]
\end{aligned}
$$

	Algorithme C_{FB} (in : *box*, inout : $[x], [\dot{x}]$)
	// Propagation
1	$[i_1] := -[x_j]$
2	$[i_2] := [x_i] + [i_1]$
3	$[i_3] := [i_2^2]$
4	$[i_4] := [-y_j]$
5	$[i_5] := [y_i] + [i_4]$
6	$[i_6] := [i_5^2]$
7	$[i_7] := [i_3] + [i_6]$
8	$[d] := [d] \cap \sqrt{[i_7]}$
	// Retro-propagation
9	$[i_7] := [i_7] \cap [d]^2$
10	$[i_3] := [i_3] \cap ([i_7] - [i_6])$
11	$[i_6] := [i_6] \cap ([i_7] - [i_3])$
12	$[i_5] := [i_5] \cap (sqr^{-1}[i_6])$
13	$[y_i] := [y_i] \cap ([i_5] - [i_4])$
14	$[i_4] := [i_4] \cap ([i_5] - [y])$
15	$[y_j] := [y_j] \cap -[i_4]$
16	$[i_2] := [i_2] \cap (sqr^{-1}[i_3])$
17	$[x_i] := [x_i] \cap ([i_2] - [i_1])$
18	$[i_4] := [i_4] \cap ([i_2] - [x_i])$
19	$[x_j] := [x_j] \cap -[i_1]$

Table C.1: Algorithm de propagation et retro-propagation appliqué à chaque mesure de distance. (2.33).

Nous obtenons alors de plus petits intervalles $[i_3] = [1, 2], [i_6] = [2, 3]$ et $[i_7] = [4, 5]$.

Le même principe peut être appliqué pour toutes les contraintes primaires de (C.20) de manière à contracter les domaines de (C.19). La séquence de contraction effectuée par l'algorithme de propagation et retro-propagation est optimale pour maximiser la contraction. L'algorithme est présenté au tableau C.1 et tourne successivement pour chaque contrainte (mesure de distance).

L'image C.3 représente successivement les contractions prenant place à chaque appel de l'algorithme.

C.3 Résolution de systèmes de satisfaction de contraintes non linéaires impliquant des fonctions dépendantes du temps

Dans le chapitre précédent nous avons présenté les notions basiques de l'analyse par intervalle et avons étudié la localisation statique d'un robot. Dans ce chapitre nous définissons les outils qui vont nous permettre de localiser dynamiquement des robots. Nous introduisons la notion de tube qui permet d'englober les trajectoires des robots à tout moment. Une arithmétique est développée autour de cette notion, et une approche par contracteurs est définie pour résoudre le problème.

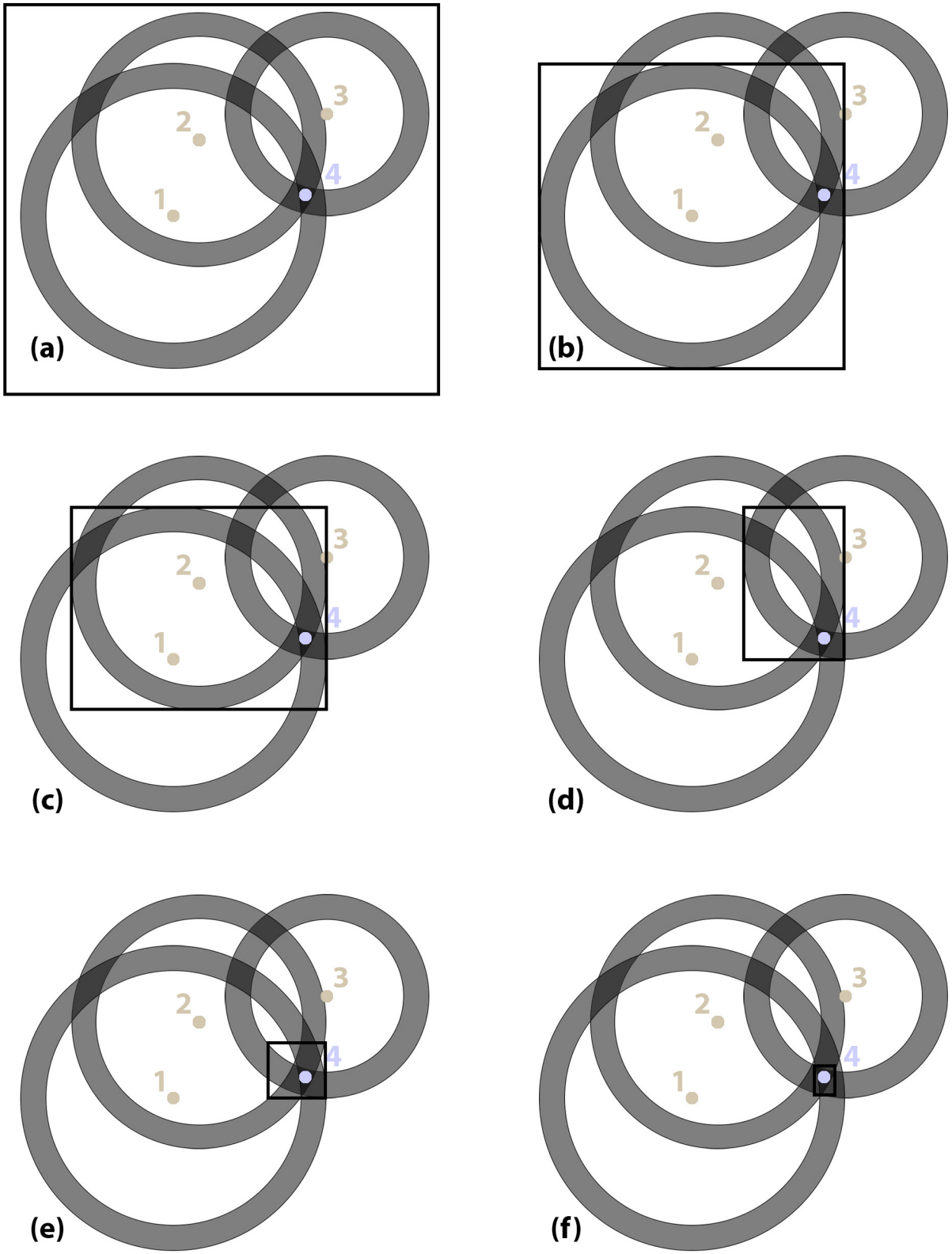

Figure C.3: Contractions successives de la boite incluant la position du robot 4 (a) initialement (b) après un appel au contracteur par propagation et retro-propagation sur la distance au robot 1, (c) au robot 2, (d) au robot 3, (e) aux robots 1, 2 et 3 une seconde fois (f) jusqu'au point fixe.

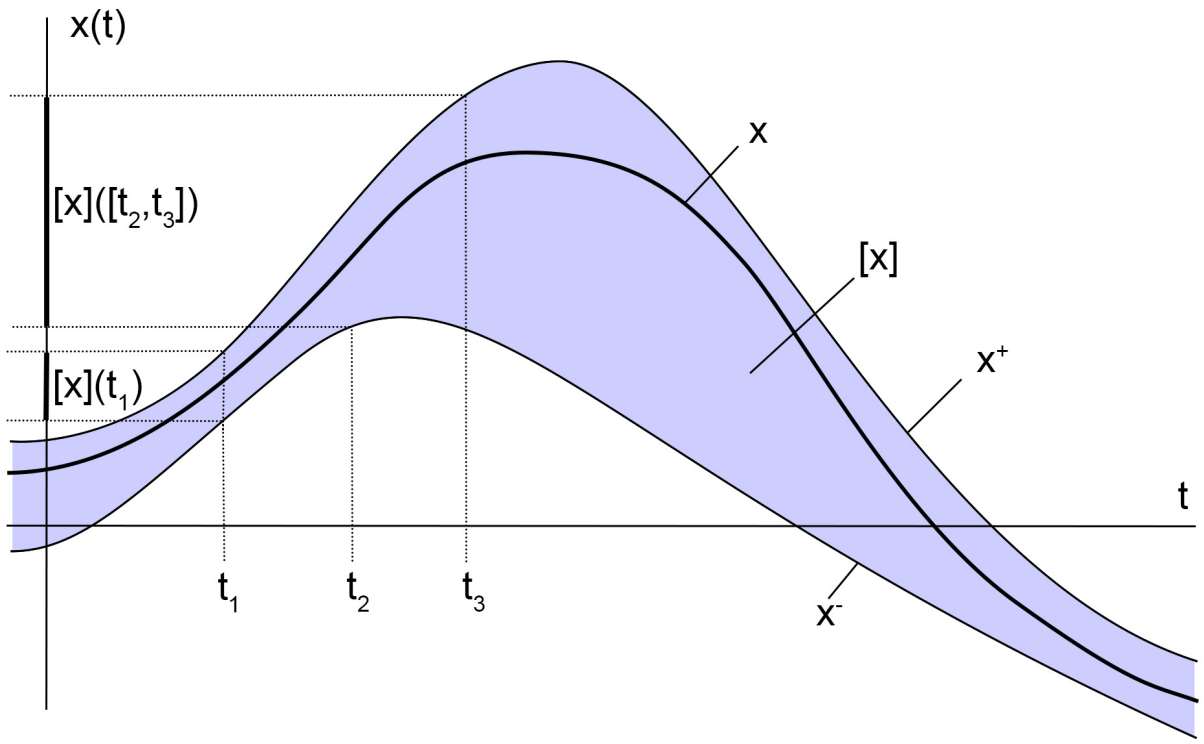

Figure C.4: Un tube $[x]$ de $\mathbb{R}$ qui encadre la fonction x.

T*ubes*

En analyse par intervalle, les variables sont généralement des booléens, des nombres entiers ou des nombres réel. L'originalité de ce chapitre est de considérer des trajectoires.

Un *tube* (ou intervalle de trajectoire) [Kurzhanski and Valyi, 1997][Milanese et al., 1996] est une vision ensembliste d'un signal aléatoire. Un tube *tube* $[\mathbf{x}]$ est un intervalle $[\mathbf{x}^-, \mathbf{x}^+]$ de l'ensemble $\mathcal{F}^n$ des fonctions de $\mathbb{R}$ dans $\mathbb{R}^n$, c'est à dire deux fonctions $\mathbf{x}^-, \mathbf{x}^+$ tel que pour tout t, $\mathbf{x}^-(t) \leq \mathbf{x}^+(t)$.

Un élément $\mathbf{x}$ de $\mathcal{F}^n$ appartient au tube $[\mathbf{x}]$ si $\forall t, \mathbf{x}(t) \in [\mathbf{x}](t)$. L'image C.4 illustre une fonction $x \in \mathcal{F}^1$ appartenant à $[x]$. Ce tube nous donne des informations sur la fonction inconnue x.

Si $\mathbf{x} \in \mathcal{F}^n$, on définit :

$$\mathbf{x}([t]) = \{\mathbf{x}(t), t \in [t]\}. \tag{C.28}$$

Numériquement, un tube $[\mathbf{x}]$ est défini par :

$$[\mathbf{x}]([t]) = \bigsqcup_{t \in [t]} [\mathbf{x}](t), \tag{C.29}$$

C'est à dire $[\mathbf{x}]([t])$ est la plus petite boite qui inclut toute boite $[\mathbf{x}](t), t \in [t]$. Il est facile de prouver que :

$$\mathbf{x} \in [\mathbf{x}], t \in [t] \Rightarrow \mathbf{x}(t) \in [\mathbf{x}]([t]), \tag{C.30}$$

et qu'aucune boite plus petite que $[\mathbf{x}]([t])$ ne satisfait cette propriété.

Ajoutons qu'il est possible de définir une arithmétique sur les tubes similaire aux intervalles. Par exemple, pour $[\mathbf{x}]$ et $[\mathbf{y}] \in \mathbb{I}\mathcal{F}^n$ et $a \in \mathbb{R}^+$

$$\begin{array}{ll} \text{(i)} & [x] + [y] = [x^- + y^-, x^+ + y^+] \\ \text{(ii)} & a[x] = [ax^- \wedge ax^+, ax^+ \vee ax^-] \end{array} \tag{C.31}$$

ou $\vee$ est la plus petite borne supérieur (ou *supremum*) et $\wedge$ est la plus grande borne inférieur (ou *infimum*). On peut aussi définir l'intégral d'un tube :

$$\int_{t_1}^{t_2} [\mathbf{x}]\,(\tau)\,d\tau = \left[\int_{t_1}^{t_2} \mathbf{x}^-\,(\tau)\,d\tau, \int_{t_1}^{t_2} \mathbf{x}^+\,(\tau)\,d\tau\right]. \tag{C.32}$$

Propagation de contrainte sur les tubes

L'arithmétique des tubes nous permet de construire des contracteurs. Considérons par exemple un signal inconnu $a(t)$ ou t est le temps. Nous souhaitons trouver le plus petit tube incluant la solution, sachant que a vérifie :

$$a([-\infty;\infty]) \subset [-1;1] \tag{C.33}$$

$$a([\frac{\pi}{2},\pi]) \subset [-0.7(t-\frac{\pi}{2})+0.99, \; -0.1(t-\frac{\pi}{2})+1.01] \tag{C.34}$$

$$a(t+\pi) = -a(t) \tag{C.35}$$

$$a(t+2\pi) = a(t) \tag{C.36}$$

$$b(t-\frac{\pi}{2}) = a(t) \tag{C.37}$$

$$b(t) = \dot{a}(t) \tag{C.38}$$

On définit initialement $[a](t) = [a^-(t), a^+(t)] = [-\infty, \infty]$ puis on applique les contracteurs associés aux contraintes C.33 et C.34. Chaque inclusion représente en réalité deux contraintes. C.34 équivaut à :

$$\forall t \in [\frac{\pi}{2},\pi], \begin{array}{l} a(t) \leqslant -0.1(t-\frac{\pi}{2})+1.01 \\ a(t) \geqslant -0.7(t-\frac{\pi}{2})+0.99 \end{array} \tag{C.39}$$

Le résultat est présenté image C.5(a). Les contraintes C.35 et C.36 montrent respectivement que le signal est symétrique par rapport au point $(\pi, 0)$ et 2π périodique. Les résultats de contraction sont présentés aux images C.5(b) et C.5(c).

Le contracteur associé à C.37 est :

$$[b(t)] = [b(t)] \cap [a(t+\frac{\pi}{2})] \tag{C.40}$$

Enfin le contracteur associé à C.38 permet d'intégrer $[b]$ pour contracter $[a]$:

$$[a(t)] = [a(t)] \cap [\int_{\tau=0}^{t} \frac{db^+(\tau)}{dt} d\tau, \int_{\tau=0}^{t} \frac{db^-(\tau)}{dt} d\tau] \tag{C.41}$$

Le résultat est illustré à l'image C.5(d). Nous pouvons réappliquer tous les contracteurs une fois (Fig. C.5(e)), trois fois (Fig. C.5(f)), cinq fois (Fig. C.5(g)) ou même jusqu'à ce que la largeur du tube soit satisfaisante ou ne contracte plus davantage (Fig. C.5(h)). Nous pouvons alors clairement reconnaitre un signal sinusoïdal.

Considérons maintenant un systeme masse-ressort présenté à l'image C.6. Dans les systèmes réels, il existe une variation de la raideur du ressort en fonction de l'élongation. Le système est alors non-linéaire et s'écrit :

$$m.\ddot{x} + \gamma.\dot{x} + \kappa.x - \beta x^3 = 0 \tag{C.42}$$

où β est la raideur du ressort, m est la masse, x est le déplacement, κ est l'élasticité du ressort γ une constante d'amortissement.

En connaissant les conditions initialles, ce système serait facilement résolvable via les méthodes standards de résolution numérique. Cependant nous considérons ici ne pas connaitre les conditions initiales. En échange nous équipons la masse d'un modem acoustique capable d'émettre une onde chaque seconde. Considérons une onde émise à t_1 voyageant à $c = 100m.s^{-1}$ vers le mur de droite, s'y réfléchissant et captée par le modem à t_2. L'équation vérifiée par la masse est alors :

$$\begin{array}{rll} (L - x(t_1)) + (L - x(t_2)) & = & c.(t_2 - t_1) \\ \Leftrightarrow x(t_2) + x(t_1) & = 2L - & c.(t_2 - t_1) \end{array} \tag{C.43}$$

A notre connaissance, ce problème ne peut être facilement résolut par les méthodes classiques. En considérant le problème comme un problème de satisfaction de contraintes, nous pouvons réécrire le système comme suit :

$$\begin{array}{ll} \frac{d}{dt}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \frac{(\beta x^2 - \kappa).x - \gamma.\dot{x}}{m} \end{pmatrix} & \text{(Equation d'évolution)} \\ x(t_i) + x(t_j) = 2L - c.(t_i - t_j) & \text{(Equation d'observation)} \end{array} \tag{C.44}$$

ou les t_j sont le temps d'émission et les t_i les temps de réception. Chaque onde acoustique peut être considéré comme une contrainte entre $[x]$ et $[\dot{x}]$. Ces contraintes permettent alors de contracter les tubes de position et de vitesse de la masse (image C.7). Notons que les équations d'observation ont généralement une forme $\mathbf{g}(\mathbf{x}(t), t) = 0$. Nous introduisons donc pour la première fois une équation d'observation inter-temporelle du type $\mathbf{g}(\mathbf{x}(t), \mathbf{x}(t'), t, t') = 0$.

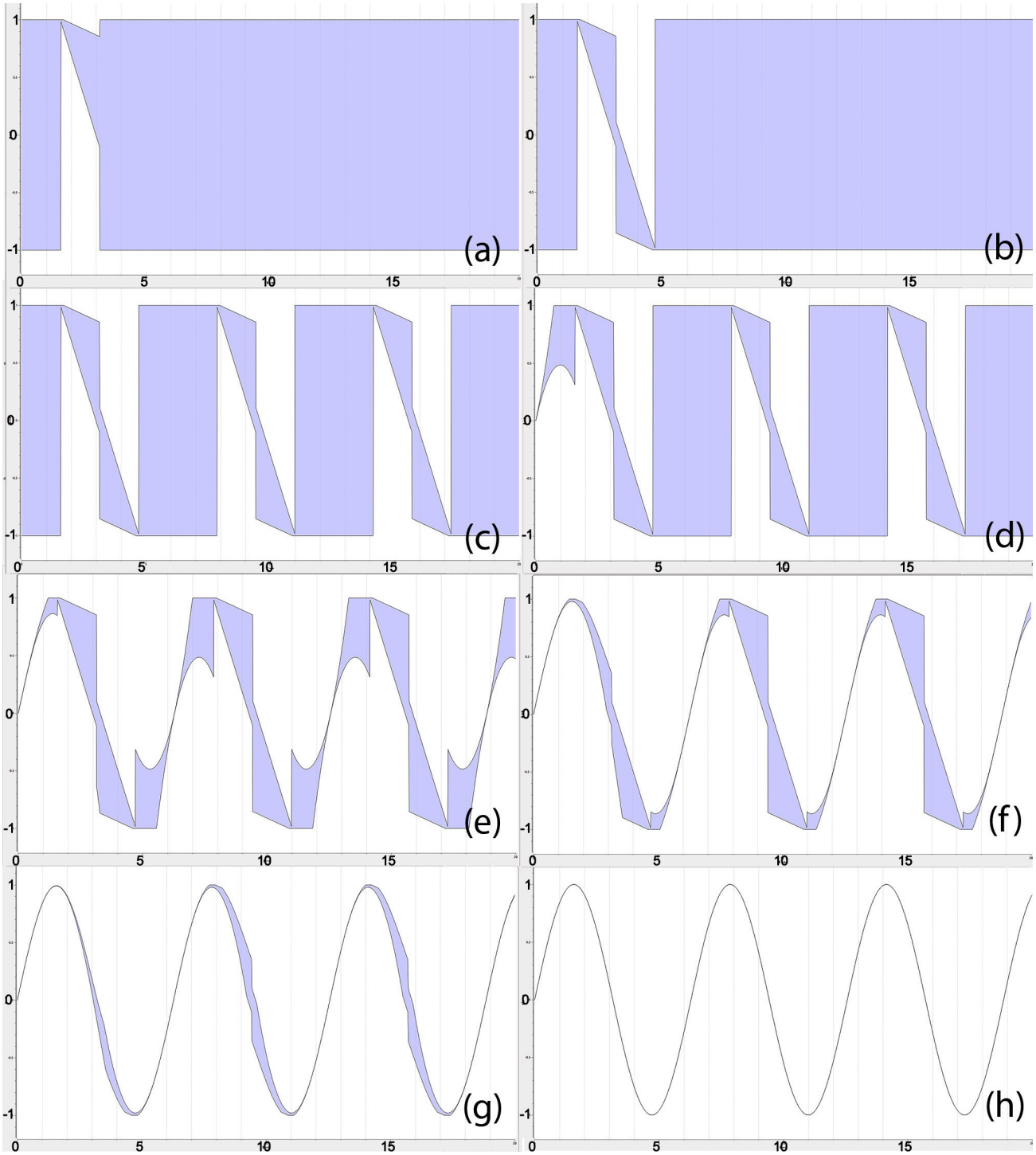

Figure C.5: Contractions successives du tube [*a*].

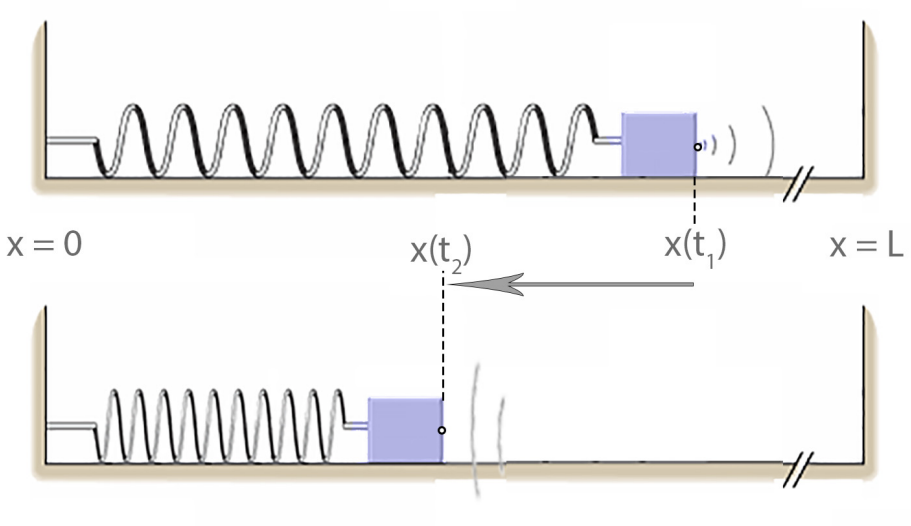

Figure C.6: Système masse-ressort-sonar

C.4 Localisation coopérative de robots sous-marins avec horloges désynchronisés

Intéressons-nous maintenant à notre problème de localisation coopérative d'AUVs en groupe. Pour un robot i dans le groupe, considérons les équations d'état suivantes :

$$\begin{array}{cc} \dot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) + \mathbf{n}_\mathbf{x} & \text{(Equation d'évolution)} \\ \mathbf{y}_i = \mathbf{g}(\mathbf{x}_i) & \text{(Equation d'observation)} \end{array} \tag{C.45}$$

ou $\mathbf{x}_i$ est le vecteur d'état du robot i, $\mathbf{u}_i$ sont ses entrés ou commandes, $\mathbf{y}_i$ ses sorties ou mesures, et $\mathbf{f}$ et $\mathbf{g}$ sont respectivement les fonctions d'évolution et d'observation. $\mathbf{n}_\mathbf{x}$ est le bruit d'état. L'incertitude sur y_i est représentée par un intervalle autour sa valeur inconnue. Notons que la fonction d'observation dépend généralement de l'état du robot au temps actuel. L'originalité de notre approche est de considérer une communication par modems acoustiques entre les robots sous forme de *ping*. Puisque les ondes sonars ne se déplacent pas instantanément, elles seront représentées par des contraintes inter-temporelles entre l'état des robots à des temps différents. Nous considérons que chaque ping transmet l'intervalle de position estimé du robot émetteur, ainsi que l'intervalle incluant l'instant d'émission du ping d'après l'horloge interne du robot émetteur.

Formalisation. Une *relation inter-temporal* (ou *ping* pour faire court) correspond à un 4-tuple $\mathbf{p} = (a, b, i, j)$ où $a \in \mathbb{R}$ est l'instant réel d'émission, $b \in \mathbb{R}$ est l'instant réel de réception, $i \in \{1, \ldots, m\}$ est le robot émetteur et $j \in \{1, \ldots, m\}$ est le robot récepteur. La causalité démontre que $b > a$. Notons $\mathbf{p}(k)$ le $k^{ème}$ ping, et t le temps réel. $\tau = h_i(t)$ est la fonction d'horloge [Srikanth and Toueg, 1987] qui à un temps réel t fait correspondre un temps local τ associé à l'horloge interne du robot i. Notons que h_i est strictement croissante si il n'y a pas de resynchronisation, sinon la fonction est croissante par morceaux.

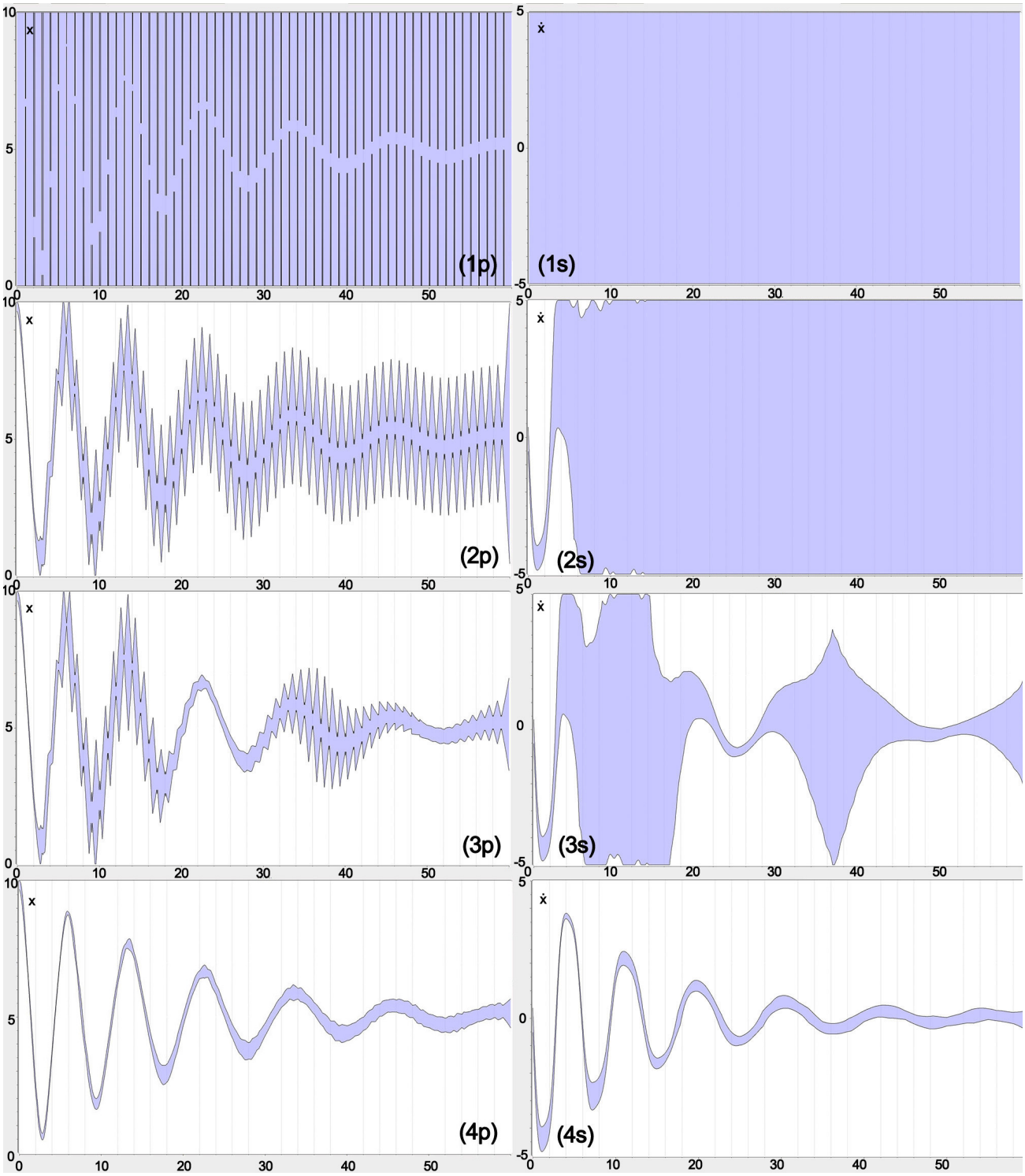

Figure C.7: Contractions successives de la position (1p, 2p, 3p et 4p) et de la vitesse (1v, 2v, 3v et 4v) de la masse.

Pour $i \in \{1,\ldots,m\}$, $t \in \mathbb{R}$ et $k \in \{1,\ldots,k_{\max}\}$, nous avons:

$$\begin{array}{ll} \text{(i)} & \dot{\mathbf{x}}_i(t) = \mathbf{f}(\mathbf{x}_i(t), \mathbf{u}_i(t)) + \mathbf{n}_{\mathbf{x}}(t) \\ \text{(ii)} & g\left(\mathbf{x}_{i(k)}(a(k)), \mathbf{x}_{j(k)}(b(k)), a(k), b(k)\right) = 0 \\ \text{(iii)} & \tilde{a}(k) = h_{i(k)}(a(k)) \\ \text{(iv)} & \tilde{b}(k) = h_{j(k)}(b(k)) \\ \text{(v)} & \dot{h}_i(t) = 1 + n_h(t) \end{array} \tag{C.46}$$

(i) correspond à la fonction d'état du i-ème robot. Le bruit d'état $\mathbf{n}_{\mathbf{x}}(t)$ est assumé borné.

(ii) est la fonction d'observation inter-temporel tel que $g : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}$ est ici :

$$g(\mathbf{x}_1, \mathbf{x}_2, a, b) = \|\mathbf{x}_1 - \mathbf{x}_2\| - c * (b - a) \tag{C.47}$$

où c est la vitesse du son dans l'eau. En utilisant une norme euclidienne, (ii) peut être réécrit ainsi :

$$c * (b(k) - a(k)) = \sqrt{\begin{array}{l}(x_{i(k)}(a(k)) - x_{j(k)}(b(k)))^2 \\ +(y_{i(k)}(a(k)) - y_{j(k)}(b(k)))^2\end{array}} \tag{C.48}$$

(iii) $\tilde{a}(k)$ correspond au temps local du robot $i(k)$ (c.à.d. l'émetteur émet le $k^{ème}$ ping).

(iv) $\tilde{b}(k)$ correspond au temps local du robot $j(k)$ (c.à.d. le récepteur reçoit le $k^{ème}$ ping).

(v) $n_h(t)$ est le bruit d'horloge et est assumé borné.

Notons que pour tout k, nous connaissons exactement: $\tilde{a}(k), \tilde{b}(k), i(k), j(k)$.

L'avantage de ce formalisme est qu'il englobe de nombreux précèdents formalismes de localisation ensembliste [Jaulin, 2011][Le Bars et al., 2010] ainsi que notre problème avec contraintes inter-temporelles sur temps incertains. A notre connaissance, un tel formalisme n'a encore jamais été proposé. Ce formalisme peut même être appliqué au problème de localisation et cartographie simultanée ($SLAM$) en considérant que les points de repère sont des robots stationnaires. Les mesures GPS en surface peuvent également être considérées comme des mesures du robot avec lui-même.

Quand un AUV 1 émet un ping k à $t = a(k)$ reçu par un autre AUV 2 à $t = b(k)$, la distance entre les deux AUVs peut etre mesuré par $c * (b(k) - a(k))$. Par conséquent, la mesure de distance est inter-temporelle, et met en relation la position de l'AUV 1 à $a(k)$ et la position de l'AUV 2 à $b(k)$. L'image C.8 illustre le principe. Notons que ces temps sont incertains puisque l'horloge des AUVs peut être désynchronisée.

Le problème de localisation coopérative décrit aux équations C.46 peut être considéré comme un problème de satisfaction de contraintes sur les tubes et nous pouvons donc définir des contracteurs associés à chaque contrainte. Pour un AUV j dans le groupe, nous pouvons contracter les tubes $[\mathbf{x}_j]$ et $[h_j]$ en utilisant le contracteur intégrale définie au chapitre précèdent. Les contracteurs associés aux contraintes (C.46i) et (C.46v) peuvent être :

$$[\mathbf{x}_j](t) = [\mathbf{x}_j](t) \cap \int_0^t ([\mathbf{f}]([\mathbf{u}_j(\tau)],[\mathbf{x}_j(\tau)]) + [\mathbf{n}_x(\tau)])d\tau \tag{C.49}$$

$$[h_j](t) = [h_j](t) \cap \int_0^t (1 + [n_h](\tau))d\tau \tag{C.50}$$

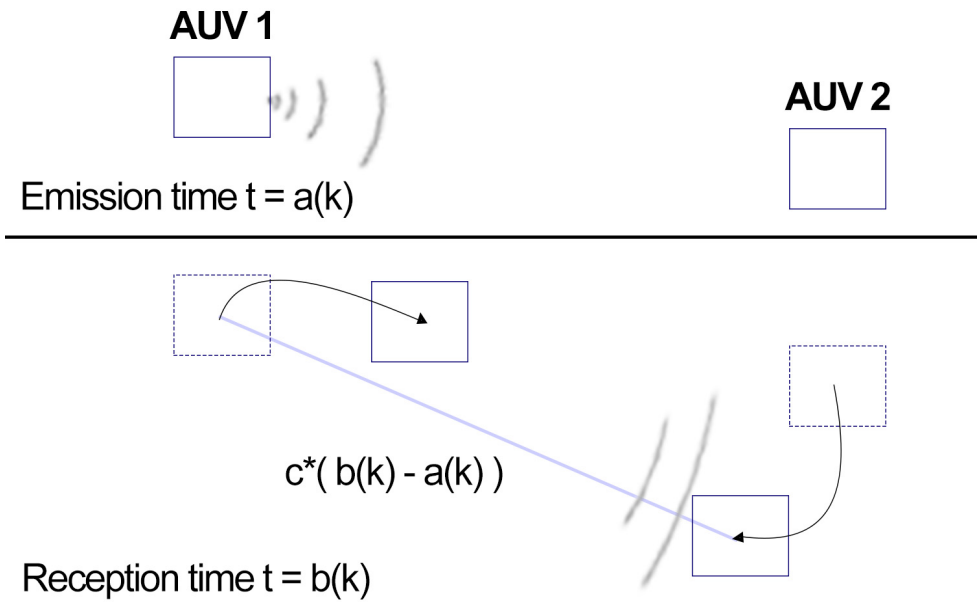

Figure C.8: Quand le robot 1 émet un ping, celui-ci est reçu par le robot 2 après $b(k)-a(k)$ seconds. Chaque robot s'est déplacé pendant ce temps. La mesure est donc entre le robot 1 à l'instant $a(k)$ et le robot 2 à l'instant $b(k)$.

Ces contracteurs sont disponibles pour tout t; et à cause du bruit d'état, les tubes $[\mathbf{x}_j]$ et $[h_j]$ croient au cours du temps. Si le robot reçoit un ping, les contraintes (C.46ii), (C.46iii) et (C.46iv) sont alors disponible pour contracter ponctuellement $[\mathbf{x}_j]$ et $[h_j]$.

Considérons un robot i émettant un ping reçu par le robot j. Chaque ping transmet la boite de position estimée $[\mathbf{x}_i]$ du robot émetteur, ainsi que l'intervalle incluant l'instant local d'émission $[a(k)]$ du ping k par le robot i d'après sa propre horloge. Le robot récepteur j recevant le ping k du robot i connait alors $[a\,(k)]$, $[b(k)]$, $[x_{i(k)}](a\,(k))$, $[x_{j(k)}](b\,(k))$, $[y_{i(k)}](a\,(k))$, $[y_{j(k)}]\,(b\,(k))$ et peut appliquer l'algorithme de propagation-rétropropagation présenté au chapitre 3 à l'équation C.48.

L'image C.9 présente les résultats de contraction des tubes de position $[\mathbf{x}_j]$ et d'horloge $[h_j]$ pour un robot $j = 4$ recevant deux pings d'autres robots. En appliquant l'algorithme de propagation et retro-propagation, les tubes sont ponctuellement contractés. En appliquant le contracteur intégrale, les tubes peuvent être contractés dans le sens des t croissants (localisation en temps réel ou online) et dans le sens des t décroissant (localisation offline prenant place plus tard après la mission).

C.5 Localisation de robots en essaim à grande échelle

Afin de démontrer l'extensibilité de notre algorithme, il est intéressant de considérer une monté en échelle de notre simulation. Afin de faire évoluer des milliers d'AUVs de manière cohérente, nous utilisons un algorithme d'essaimage proposé par *Craig Reynolds* en 1987 [Reynolds, 1987], basé sur trois règles : *séparation*, *cohésion* and *alignement*. Chaque règle détermine comment chaque membre de l'essaim doit réagir par rapport aux autres membres dans son *voisinage*. Les membres en dehors d'une certaine distance d de

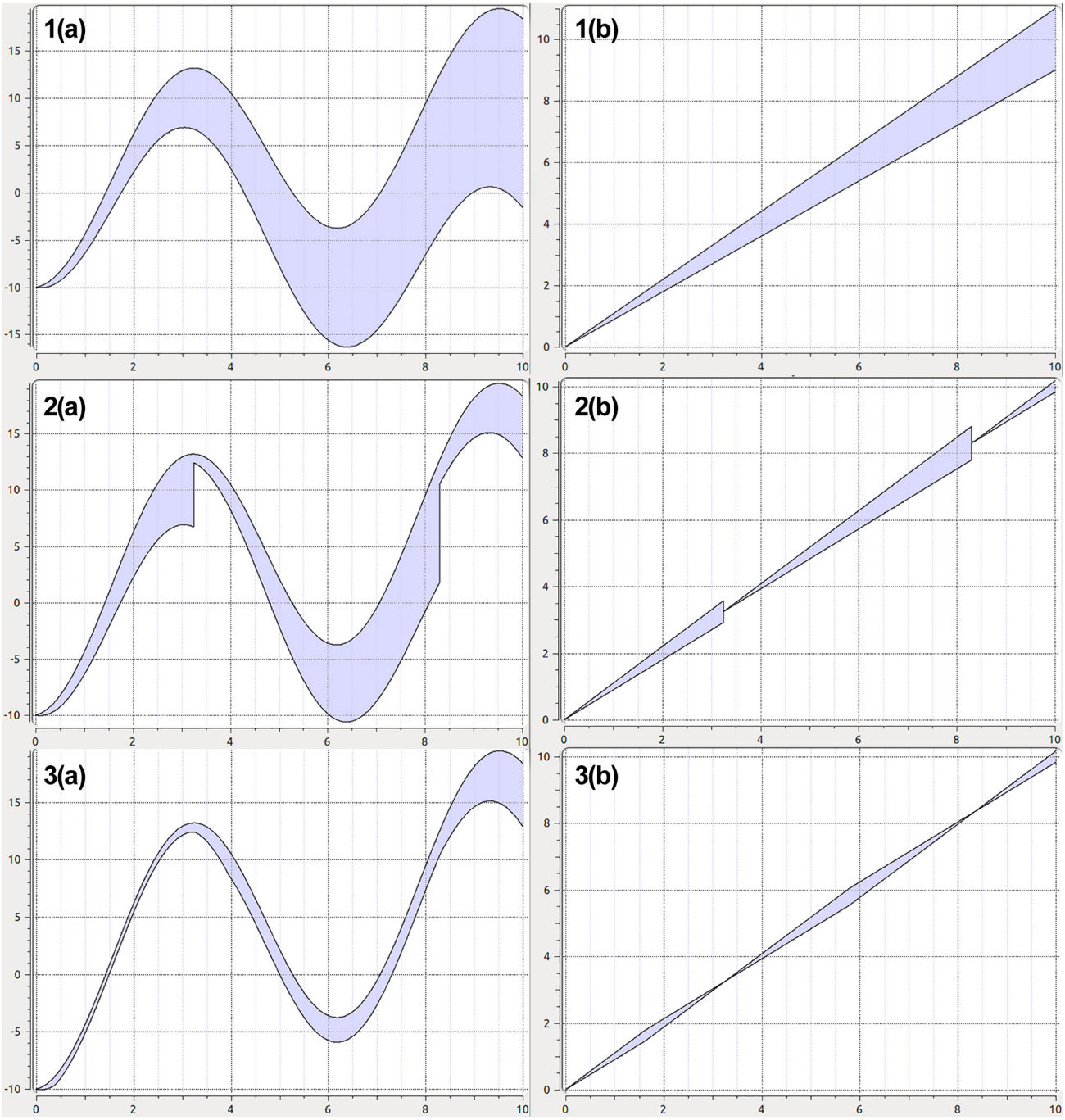

Figure C.9: Tubes contractés pour (a) $[h_4]$ et (b) $[x_4]$ si (1) aucun ping n'est reçu, (2) deux pings sont reçus avec propagation online, (3) deux pings sont reçus avec propagation offline.

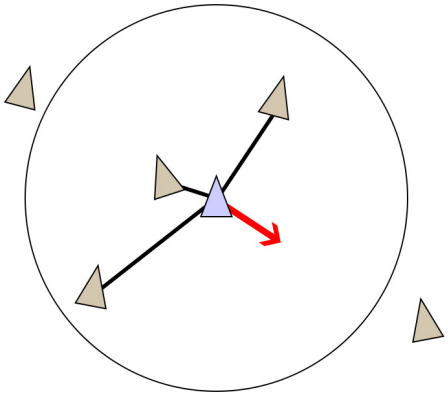

Figure C.10: Illustration de la séparation.

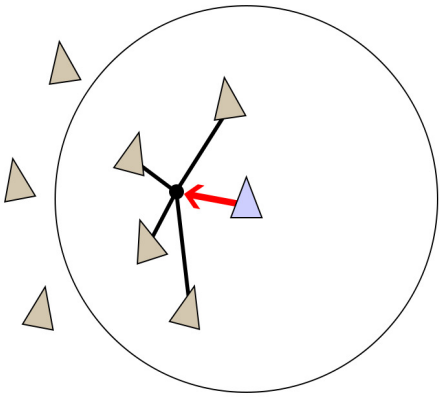

Figure C.11: Illustration de la cohésion

voisinage sont ignorés. Dans notre cas, d sera la portée du modem acoustique utilisé.

· La séparation est le principe d'éloigner l'AUV de ses proches voisins. Cela permet d'éviter les collisions.

· La cohésion est le principe de rapprocher l'AUV du centre de masse de ses voisins. La cohésion et la séparation travaille en opposition afin de maintenir un comportement de groupe.

· L'alignement dirige l'AUV dans la direction moyenne de ses voisins.

L'*essaimage* est le principe d'évoluer en essaim et peut être réalisé en additionnant la séparation, la cohésion and l'alignement. Le mouvement obtenu est alors très proche du comportement des essaims d'oiseaux.

Nous appliquons alors notre algorithme de localisation établie aux chapitres précédents. La simulation démontre alors qu'il est possible de monter en échelle jusqu'à 1,000 AUVs tout en restant en temps-réel sur un 3.2GHz core i7. Notons que de plus notre algorithme a été dévellopé pour être distribué sur chaque AUV sans la gérance d'un système central.

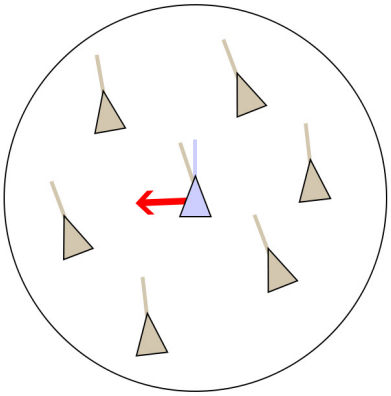

Figure C.12: Illustration de l'alignement.

C.6 Conclusion

La localisation d'AUVs en essaim est un problème difficile. Au chapitre 1, nous avons introduit le sujet, puis au chapitre 2, nous avons présenté l'analyse par intervalle et son application à la localisation en robotique mobile. Le chapitre 3 a étendu la notion de contracteur et a proposé l'évaluation de contracteurs minimaux par transformations géométriques. Le chapitre 4 à formalisé le problème de localisation dynamique d'AUVs sous la forme d'un problème de satisfaction de contraintes inter-temporel, et a proposé un nouvel outil, le tube, pour résoudre ce problème. Le chapitre 5 s'est ensuite intéressé à un modèle d'AUV plus proche de la réalité, pour lequel l'horloge du robot est désynchronisée par rapport au temps réel. Enfin le chapitre 6 conclu la thèse par une forte monté en échelle des algorithmes et simulations présentés dans les chapitres précédent.

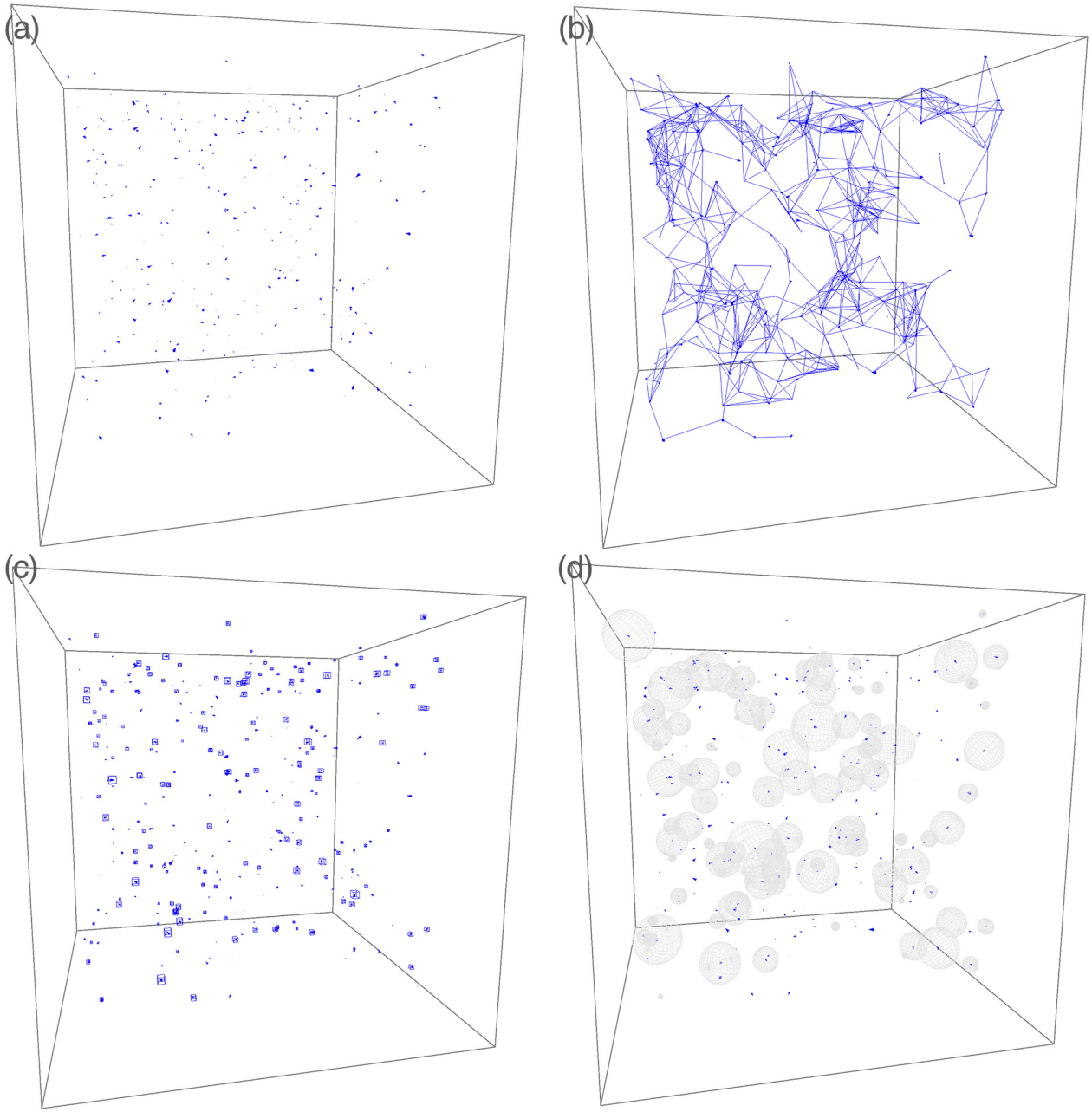

Figure C.13: Simulation de 500 AUVs. (a) représente la position réel des AUVs. (b) montre un lien entre les AUVs à porté de modem acoustique, (c) montre les boites d'incertitude sur la position des AUVs et (d) illustre la propagation des pings.

Articles, Congresses and Reports

Journal articles

A. Bethencourt and L. Jaulin. 3D reconstruction using the Kinect sensor coupled with an IMU.
Published in *InTech, International Journal of Advanced Robotics*, 2012.

A. Bethencourt and L. Jaulin. Cooperative localization of underwater robots with unsynchronized clocks.
Published in *Paladyn, International Journal of Behavioral Robotics, Special issue on underwater robotics, VERSITAS*, Volume 4, Issue 4, pp 233-244, 2013.

A. Bethencourt and L. Jaulin. Solving non-linear constraint satisfaction problems involving time-dependant functions.
Published in *Mathematics in Computer Science, Special Issue on Interval Methods and Applications* , 2014.

A. Stancu, A. Bethencourt and L. Jaulin. Building an optimal contractor for atan2 and its application to robot localization.
Submitted to Reliable Computing, 2014.

A. Bethencourt and L. Jaulin. Large-scale swarm localization using interval analysis.
Submitted to Swarm Intelligence, 2014.

Congresses with a selection committee

O. Menage, A. Bethencourt, P. Rousseaux, S. Prigent. VAIMOS : Realization of an autonomous robotic sailboat.
In *Proceedings of the 6th International Robotic Sailing Conference, IRSC-WRSC 2013*, IRSC-WRSC 2013, France, pp 25-36, 2013.

Congresses without a selection committee

A. Bethencourt and L. Jaulin. 3D reconstruction using the Kinect sensor coupled with an IMU.
In *Small Worskshop on Interval Methods*, *SWIM 2012*, Germany, 2012.

A. Bethencourt and L. Jaulin. Cooperative localization of underwater robots with unsynchronized clocks.
In *Small Worskshop on Interval Methods, SWIM 2013*, France, 2013.

A. Bethencourt and L. Jaulin. Solving non-linear constraint satisfaction problems involving time-dependant functions.
In *Méthodes ensemblistes pour l'automatique*, *GDR MACS*, France, 2013.

A. Bethencourt and L. Jaulin. Introducing Interval Analysis and their applications to robotics.
In *Aeorospace research committee*, University of Manchester, 2014.

L. Jaulin and A. Bethencourt. Nonlinear state estimation with delays.
In *1st Small Symposium on Set-Membership: Applications, Reliability and Theory*, University of Manchester, Aerospace Research Institute, UK, 2014

L. Jaulin and A. Bethencourt. Solving geometrical constraints in space-time.
In *Contraintes et géométrie*, Université de Nantes, France, 2014

Other documents

A. Bethencourt. Progress report for *MRIS/DGA* and *EDSICMA*, 2012.

A. Bethencourt. Realization report for *Coupe de France de Robotique, E=M6*, 2012.

F. Le Bars, Y. Sliwka, A. Bethencourt, Realization report for *SAUC-E*, 2012.

A. Bethencourt. Progress report for *MRIS/DGA* and *EDSICMA*, 2013.

A. Bethencourt. Realization report for *Coupe de France de Robotique, E=M6*, 2013.

F. Le Bars, A. Bethencourt, Realization report for *SAUC-E*, 2013.

A. Bethencourt. Mid-thesis progress defense with Luc Jaulin, Laurent Hardoin, Eva Crück and Gilles Charbert, 2013.

A. Bethencourt. Progress report for *MRIS/DGA* and *EDSICMA*, 2014.

Summary of main contributions

My main contributions :

- Formalization of the swarm localization problem. Proposed a solution based on interval analysis.
- Development of the arithmetics of tubes. Proved multiple propositions involving tubes and time-dependant functions.
- Formalization of the clock synchronization problem in a swarm. Proposed a solution based on tube arithmetic.
- Proposed a method for solving inter-temporal constraint satisfaction problem with uncertain times.
- Integration of tubes and their arithmetic into the *IBEX* library.
- Integration of an atan2 contractor into the *IBEX* library using geometrical transformations.
- Development of *SwarmX*, an underwater swarm simulator using tube arithmetic.
- Software and hardware development for the 2012 and 2013 *SAUC-E* underwater robotics competition.
- Software and hardware development for the 2012, 2013 and 2014 *Coupe de France de Robotique*.
- Software and hardware development for the 2010 and 2011 *DGA CAROTTE* competition.

Additional activities

During my Ph.D., I had the chance to preside the robotics club and managed a team of students in three robotics competitions : *Coupe de France de Robotique*, *SAUC-E* and *CAROTTE*. The goal in all these competitions was to build robots from scratch. We had to develop the hardware along with the software. In all three competitions, one of the main difficulty was to localize the robots as we couldn't use the *Global Positioning System* (*GPS*), either because the robots were indoor or underwater. The interesting point is that we used different approaches in each competition, both probabilistic and membership, illustrating perfectly the state of the art in robot localization.

Coupe de France de Robotique

The *Coupe de France de Robotique* (or French Robotics Cup) involves 200 teams from universities and schools in Europe. It takes place at *La Ferté Bernard* during 4 days in June. The competition consists of 5 matches of 90 seconds against a robots from other schools on an arena of 3 by 2 meters filled with objects. Objectives are usually to collect the objects and push buttons on the arena. For this, a good localization on the arena is essential. Moreover the robots have to avoid collisions with each other and with the environment. Therefore we chose to equip our robot with scanning range finders to measure distances around the robot, and especially to measure distances to the known borders of the arena. Our approach for this competition particularly emphasize the state of the art in probabilistic localization as we used an *Affine Monte-Carlo localization* algorithm (*AMCL*) embedded into *ROS* (*Robot Operating System*), a famous middleware for robotics. *Monte Carlo* localization, also known as particle filter localization, is an algorithm for robots to localize themselves [Thrun et al., 2005][Dellaert et al., 1999][Thrun et al., 2000]. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, i.e. a hypothesis of where the robot is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning that the robot has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive *Bayesian* estimation, i.e. how well the actual sensed data correlate with the predicted state. Ultimately, the particles should converge towards the actual pose of the robot. Fig. C.14 highlights the competition.

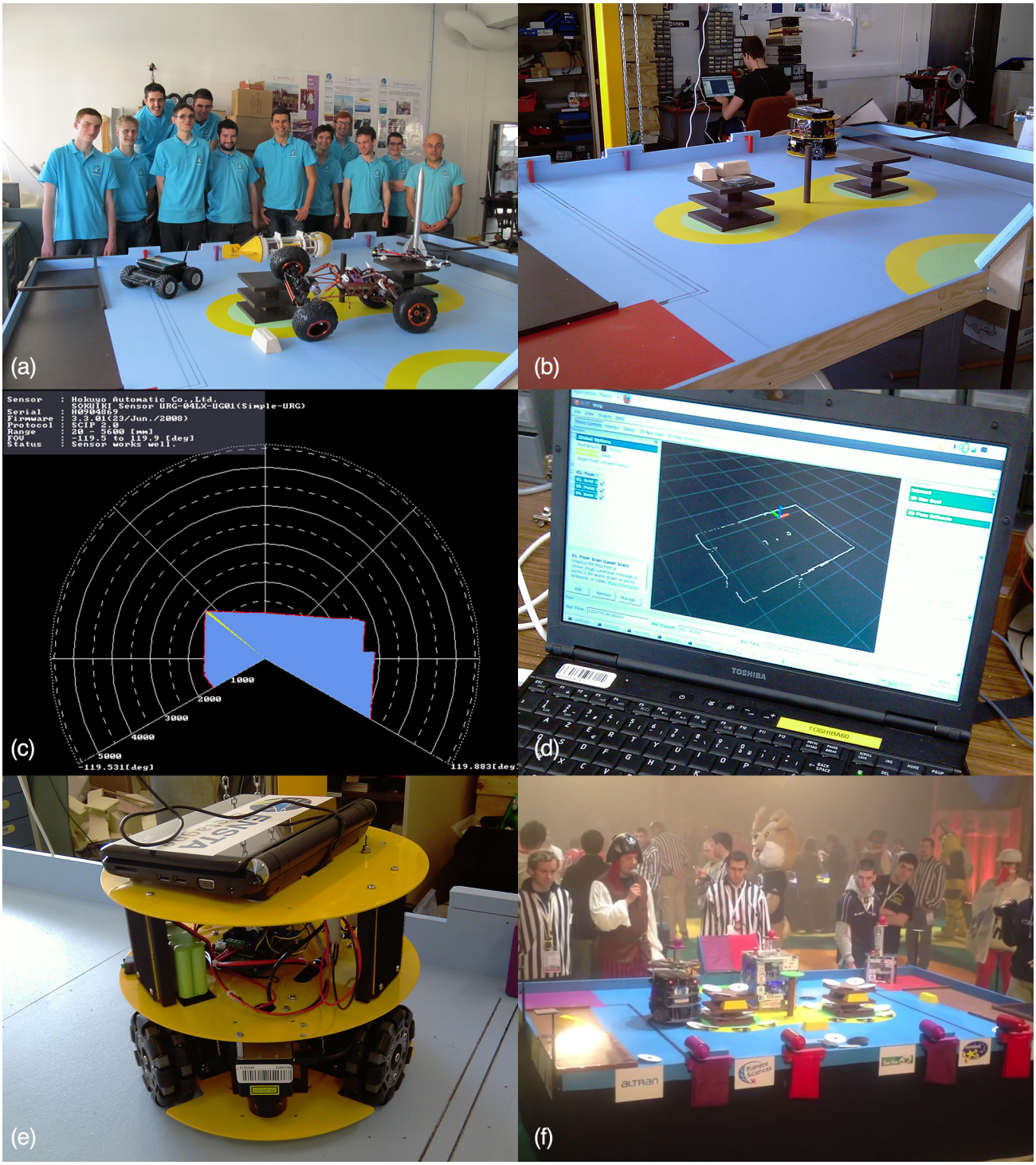

Figure C.14: (a) The ENSTA robotics club. (b) Reproduction of the competition arena of the *Coupe de France de Robotique*. (c) Measures provided in real-time by the scanning range finder. (d) Localization of the robot in the arena, computed by *ROS*'s *AMCL*. (e) The early version of the *ENSTA* robot. On top level is the computer running *ROS*. In the middle level is an *Arduino* card with power bridges to control the motors, and on the lower level are the scanning range finders and the wheels. (f) The latest version of the *ENSTA* robot in the competition against the *SUPELEC* robot.

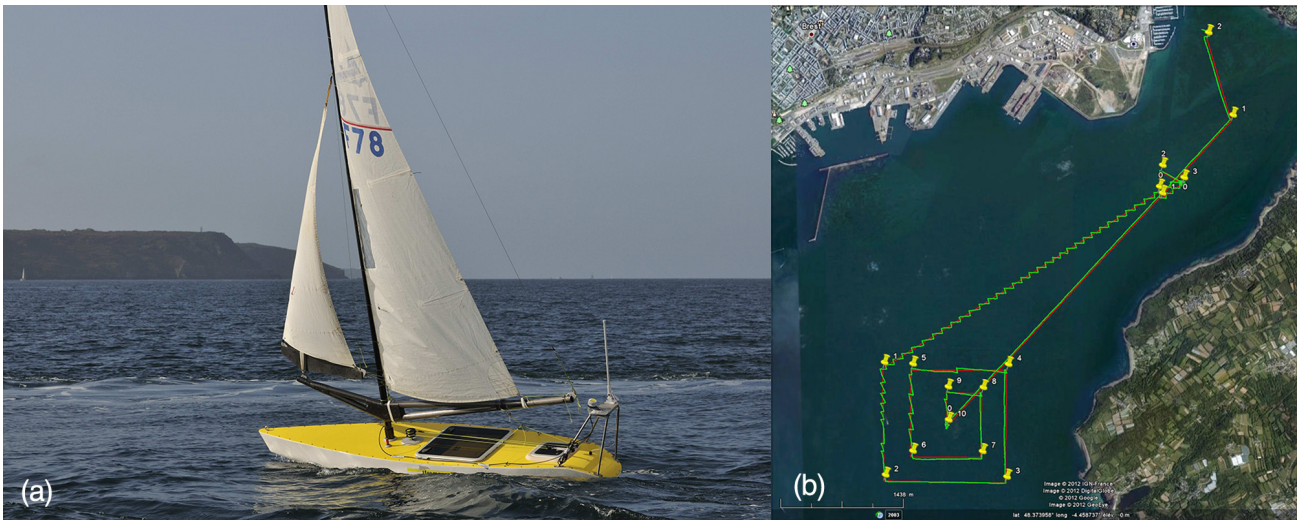

Figure C.15: (a) The VAIMOS sailboat at the WRSC. (b) In red is the requested trajectory and in green is the actual trajectory of the robot.

WRSC

The *WRSC* (World Robotics Sailing Championship) involves sailing robots from all around the world. The goal is to race against others given requested GPS checkpoints, and detect various objects along the way. Obviously, the robots are completely autonomous and must sail with the wind conditions in real-time. To localize themselves, the robots can here use the GPS.

Although I did not take part in the competition itself, I worked on the proceedings [Menage et al., 2014] and presented the construction of the ENSTA/Ifremer sailboat *VAIMOS (Voilier Autonome Instrumenté pour Mesures Océanographiques de Surface).*

CAROTTE

The *CAROTTE (CArtographie RoboTique TErrestre)* competition was organized by the *DGA* for military applications. The goal was for a team of robots to map an indoor environment in an autonomous and efficient way, while avoiding and identifying various objects in the arena. Presented in Appendix B, I realized a visual goniometric system for the robots to localize themselves in respect to each other.

SAUC-E

The objective of the *SAUC-E (Student Autonomous Underwater Challenge - Europe)* competition was to design and build an autonomous underwater vehicle capable of performing realistic missions. Various objectives were presented, among them: getting through an underwater gate, detecting a pipeline, a buoy and a beacon, and following a wall. The main difficulty was obviously to work in an underwater environment. Waterproofing was a big concern, and the localization was done based on acoustic waves echoing from the walls of the harbour. An interval method proposed by [Sliwka, 2011] was used. Fig. C.17 highlights the

Figure C.16: (a) The *Université d'Angers* team at the CAROTTE competition with its 3 robots ready to start the mapping mission. (b) The arena, buit to reproduce an indoor environment.

competition. Notice that I used the same robots in the 5th chapter of this thesis to test my own algorithms.

For more info, please visit http://aymericbethencourt.com/wordpress/robotics/.

Teaching

Over the 3 years of my Ph.D., I also teached a total of 120 hours of graduate level classes. I was mainly involved in :

· Computer Vision, for which I had my own class. I personally wrote the course, presentation, and associated documents, teached and designed coding exercises for the labs. I also wrote, gave and corrected the exam on the subject.

· Automatics and robotics, for which I was mainly involved in teaching the labs and occasionally the course.

· Robot Operating System, for which I designed and teached a lab.

· C++ Programming and Qt Integration, for which I designed and teached a lab.

· Java Programming, for which I teached multiple labs.

For more info, please visit http://aymericbethencourt.com/wordpress/teaching/.

Figure C.17: (a) The ENSTA CISSEAU team at the SAUC-E competition, (b) getting a robot into the water, (c) and launching the mission.

Videos

- Personal interview on national television France 2

http://youtu.be/Zwjbufay9Z0

- *SwarmX v1* : Simulation of a group of 6 AUVs with instantaneous communication

http://youtu.be/0cjzzsaWTvA

- *SwarmX v2* : Simulation of a group of 6 AUVs with inter-temporal communication and unsynchronized clocks

http://youtu.be/7Uzjr-U7xY4

- *SwarmX v3* : Simulation of a swarm of 1,000 AUVs with inter-temporal communication and unsynchronized clocks

http://youtu.be/F4ntGSBS1J4

- Mass-Spring-Sonar simulation

http://youtu.be/57E8k0q9YIU

- Sea testing of our algorithms using 2 ENSTA AUVs

http://youtu.be/1QFpko0tYO0

- 3D reconstruction of my own car using interval analysis on the *Kinect* device

Reconstruction : http://youtu.be/HKuSv8X3UWM

Captures :http://youtu.be/GZHYMGErA6E

- The ENSTA team at the 2012 Coupe de France de Robotique

Part 1 : http://youtu.be/8TqE9YJjbwQ

Part 2 : http://youtu.be/YDg8NnFKXSo

- S.W.I.M. 2012 : 3D reconstruction using the Kinect sensor coupled with an IMU

http://youtu.be/HlHMhkBT77w

- S.W.I.M. 2013 : Cooperative localization of underwater robots with unsynchronized clocks

Part 1 : http://youtu.be/8DgbvWDzFGw

Part 2 : http://youtu.be/_xl-F0ckkFI

Index

Bibliography

[Abdallah et al., 2008] Abdallah, F., Gning, A., and Bonnifait, P. (2008). Box particle filtering for nonlinear state estimation using interval analysis. *Automatica*, 44(3):807–815.

[Araya et al., 2008] Araya, I., Neveu, B., and Trombettoni, G. (2008). Exploiting Common Subexpressions in Numerical CSPs. In *Proc. CP, Constraint Programming*, pages 342–357, LNCS 5202.

[Aubin and Frankowska., 1990] Aubin, J. and Frankowska., H. (1990). *Set-Valued Analysis*. Birkhäuser, Boston.

[Aubry et al., 2013] Aubry, C., Desmare, R., and Jaulin, L. (2013). Loop detection of mobile robots using interval analysis. *Automatica*, 49(2):463–470.

[Baccou and Jouvencel, 2002] Baccou, P. and Jouvencel, B. (2002). Homing and navigation using one transponder for auv, post-processing comparisons results with long base-line navigation. *IEEE Int. Conf. Robotics and Automation*, pages 11–15.

[Bahr et al., 2006] Bahr, A., Leonard, J. J., Bahr, A., and Leonard, J. J. (2006). Cooperative localization for autonomous underwater vehicles. In *IN PROC. 10TH INTERNATIONAL SYMPOSIUM ON EXPERIMENTAL ROBOTICS (ISER), RIO DE*.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Gool, L. (2006). Surf : Speeded up robust features. pages 404–417.

[Bekris et al., 2006] Bekris, K., Click, M., and Kavraki, E. (2006). Evaluation of algorithms for bearing-only slam. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1937–1943.

[Benhamou et al., 1999] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J. F. (1999). Revising hull and box consistency. In *Proceedings of the International Conference on Logic Programming*, pages 230–244, Las Cruces, NM.

[Berz and Makino, 1998] Berz, M. and Makino, K. (1998). Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4(3):361–369.

[Besl and McKay, 1992] Besl, P. and McKay, H. D. (1992). A method for registration of 3 -d shapes. *Pattern Analysis and Machine Intelligence*, 14(2):239–256.

[Bishop et al., 2009] Bishop, A., Anderson, B. D. O., Fidan, B., Pathirana, P., and Mao, G. (2009). Bearing-only localization using geometrically constrained optimization. *Aerospace and Electronic Systems, IEEE Transactions on*, 45(1):308–320.

[Bonnifait and Garcia, 1996] Bonnifait, P. and Garcia, G. (1996). A multisensor localization algorithm for mobile robots and its real-time experimental validation. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1395–1400. IEEE.

[Bonnifait and Garcia, 1998] Bonnifait, P. and Garcia, G. (1998). Design and experimental validation of an odometric and goniometric localization system for outdoor robot vehicles. *IEEE Transactions on robotics and automation*, 14(4):541–548.

[Bouissou et al., 2013] Bouissou, O., Chapoutot, A., and Djoudi, A. (2013). Enclosing temporal evolution of dynamical systems using numerical methods. In *5th NASA Formal Methods Symposium, NFM 2013*, NASA Ames Research Center, Moffett Field, CA, USA.

[Bouron, 2002] Bouron, P. (2002). *Méthodes ensemblistes pour le diagnostic, l'estimation d'état et la fusion de données temporelles.* PhD dissertation, Université de Compiègne, Compiègne, France.

[Brambilla et al., 2013] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

[Bürkle et al., 2011] Bürkle, A., Segor, F., and Kollmann, M. (2011). Towards autonomous micro uav swarms. *Journal of intelligent & robotic systems*, 61(1-4):339–353.

[Ceberio and Granvilliers, 2001] Ceberio, M. and Granvilliers, L. (2001). Solving nonlinear systems by constraint inversion and interval arithmetic. In *Artificial Intelligence and Symbolic Computation*, volume 1930, pages 127–141, LNCS 5202.

[Chabert and Jaulin, 2009] Chabert, G. and Jaulin, L. (2009). Contractor Programming. *Artificial Intelligence*, 173:1079–1100.

[Combastel, 2005] Combastel, C. (2005). State bounding observer for uncertain non-linear continuous-time systems based on zonotopes. In *Proc. of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, pages 7228–7234.

[Couzin et al., 2005] Couzin, I. D., Krause, J., Franks, N. R., and Levin, S. A. (2005). Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516.

[Delanoue et al., 2006] Delanoue, N., Jaulin, L., and Cottenceau, B. (2006). Using interval arithmetic to prove that a set is path-connected. *Theoretical Computer Science, Special issue: Real Numbers and Computers*, 351(1):119–128.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, A., and Thrun, S. (1999). Monte-Carlo localization for mobile robots. In *"Proceedings of the IEEE International Conference on Robotics and Automation"*, pages 1322–1328, Detroit, Michigan.

[Di Marco et al., 2001] Di Marco, M., Garulli, A., Lacroix, S., and Vicino, A. (2001). Set membership localization and mapping for autonomous navigation. *International Journal of Robust and Nonlinear Control*, 7(11):709–734.

[Dong et al., 2013] Dong, H., Zhao, Y., and Gao, S. (2013). A fuzzy-rule-based couzin model. *Journal of Control Theory and Applications*, 11(2):311–315.

[Drevelle, 2011] Drevelle, V. (2011). *Etude de mÃlthodes ensemblistes robustes pour une localisation multisensorielle intÃÍgre. Application Ãă la navigation des vÃlhicules en milieu urbain.* PhD dissertation, UniversitÃl' de Technologie de CompiÃÍgne, CompiÃÍgne, France.

[Drevelle and Bonnifait, 2009] Drevelle, V. and Bonnifait, P. (2009). High integrity gnss location zone characterization using interval analysis. In *ION GNSS*.

[Drocourt et al., 2005] Drocourt, C., Delahoche, L., E. Brassart, B. M., and Clerentin, A. (2005). Incremental construction of the robot's environmental map using interval analysis. *Global Optimization and Constraint Satisfaction: Second International Workshop, COCOS 2003*, 3478:127–141.

[Engelharda et al., 2011] Engelharda, N., Endresa, F., Hessa, J., Sturmb, J., and Burgard, W. (2011). Real-time 3d visual slam with a hand- held rgb - d camera.

[Fischler and Bolles, 1987] Fischler, M. and Bolles, R. (1987). Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography. *Graphics and Image Processing*, 24(6).

[Gning and Bonnifait, 2005] Gning, A. and Bonnifait, P. (2005). Dynamic vehicle localization using constraints propagation techniques on intervals a comparison with kalman filtering. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4144–4149. IEEE.

[Gning and Bonnifait, 2006] Gning, A. and Bonnifait, P. (2006). Constraints propagation techniques on intervals for a guaranteed localization using redundant data. *Automatica*, 42(7):1167–1175.

[Gollamudi et al., 1996] Gollamudi, S., Nagaraj, S., Kapoor, S., and Huang, Y.-F. (1996). Set-membership state estimation with optimal bounding ellipsoids. In *Int. Symposium on Information Theory and its Applications*.

[Grisetti et al., 2010] Grisetti, G., Kummerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010). Hierarchical optimization on manifolds for online 2d and 3d mapping. *In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), Anchorage, AK, USA*.

[Grossman et al., 2008] Grossman, D., Aranson, I., and Jacob, E. B. (2008). Emergence of agent swarm migration and vortex formation through inelastic collisions. *New Journal of Physics*, 10(2):023036.

[Guyonneau, 2013] Guyonneau, R. (2013). *MÃlthodes ensemblistes pour la localisation en robotique mobile.* PhD dissertation, UniversitÃl' d'Angers, Angers, France.

[Hauert et al., 2011] Hauert, S., Leven, S., Varga, M., Ruini, F., Cangelosi, A., Zufferey, J.-C., and Floreano, D. (2011). Reynolds flocking in reality with fixed-wing robots: communication range vs. maximum turning rate. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 5015–5020. IEEE.

[Henry et al., 2010] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. *In Proc. of the Intl. Symp. on Experimental Robotics*.

[Herrero, 2006] Herrero, P. (2006). *Quantified Real Constraint Solving Using Modal Intervals with Applications to Control*. PhD dissertation, Universitat de Girona, Girona,Spain.

[Hodgins and Brogan, 1994] Hodgins, J. and Brogan, D. (1994). Robot herds: Group behaviors for systems with significant dynamics. In *Proceedings of Artificial Life IV*, pages 319–324.

[Izadi et al., 2011] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. (2011). Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA. ACM.

[Jaulin, 2002] Jaulin, L. (2002). Nonlinear bounded-error state estimation of continuous-time systems. *Automatica*, 38:1079–1082.

[Jaulin, 2011] Jaulin, L. (2011). Range-only SLAM with occupancy maps; A set-membership approach. *IEEE Transaction on Robotics*, 27(5):1004–1010.

[Jaulin et al., 2001a] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001a). Applied interval analysis. *Springer-Verlag*.

[Jaulin et al., 2001b] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001b). *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, London.

[Jaulin et al., 2002] Jaulin, L., Kieffer, M., Walter, E., and Meizel, D. (2002). Guaranteed robust nonlinear estimation with application to robot localization. *IEEE Transactions on systems, man and cybernetics; Part C Applications and Reviews*, 32(4):374–382.

[Jaulin and Walter, 1993] Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064.

[Jaulin et al., 1996] Jaulin, L., Walter, E., and Didrit, O. (1996). Guaranteed robust nonlinear parameter bounding. In *Proceedings of CESA'96 IMACS Multiconference (Symposium on Modelling, Analysis and Simulation)*, volume 2, pages 1156–1161, Lille, France.

[Kurzhanski and Valyi, 1997] Kurzhanski, A. and Valyi, I. (1997). *Ellipsoidal Calculus for Estimation and Control*. Birkhäuser, Boston, MA.

[Kushleyev et al., 2013] Kushleyev, A., Mellinger, D., Powers, C., and Kumar, V. (2013). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300.

[Lacroix et al., 2002] Lacroix, S., Mallet, A., Bonnafous, D., Bauzil, G., Fleury, S., Herrb, M., and Chatila, R. (2002). Autonomous rover navigation on unknown terrains: Functions and integration. *The International Journal of Robotics Research*, 21(10-11):917–942.

[Le Bars, 2011] Le Bars, F. (2011). *Analyse par intervals pour la localisation et la cartographie simultanÃľes; Application Ãă la robotique sous-marine*. PhD dissertation, ENSTA Bretagne, Brest, France.

[Le Bars et al., 2010] Le Bars, F., Bertholom, A., Sliwka, J., and Jaulin, L. (2010). Interval slam for underwater robots; a new experiment. In *NOLCOS 2010*, Italy.

[Lemaire et al., 2007] Lemaire, T., Berger, C., Jung, I.-K., and Lacroix, S. (2007). Vision-based slam: Stereo and monocular approaches. *International Journal of Computer Vision*, 74(3):343–364.

[Lemaire and Lacroix, 2007] Lemaire, T. and Lacroix, S. (2007). Slam with panoramic vision. *Journal of Field Robotics*, 24(1-2):91–111.

[Lemaire et al., 2005] Lemaire, T., Lacroix, S., and Sola, J. (2005). A practical 3d bearing-only slam algorithm. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2449–2454. IEEE.

[Leonard and Durrant-Whyte, 1992] Leonard, J. and Durrant-Whyte, H. (1992). Dynamic Map Building for an Autonomous Mobile Robot. *International Journal of Robotics Research*, 11(4).

[Lhommeau et al., 2004] Lhommeau, M., Hardouin, L., Cottenceau, B., and Jaulin, L. (2004). Interval analysis and dioid: Application to robust controller design for timed event graphs. *Automatica*, 40(11):1923–1930.

[Lin et al., 2005] Lin, Y., Vernaza, P., Ham, J., and Lee, D. (2005). Cooperative relative robot localization with audible acoustic sensing. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3764–3769.

[Ljung, 1979] Ljung, L. (1979). Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Systems*, 24:36–50.

[Logothetis et al., 1997] Logothetis, A., Isaksson, A., and Evans, R. (1997). An information theoretic approach to observer path design for bearings-only tracking. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 4, pages 3132–3137 vol.4.

[Lowe, 1999] Lowe, D. (1999). Proceedings of the seventh ieee international conference on computer vision. 2:1150–1157.

[Maksarov and Norton, 1996] Maksarov, D. and Norton, J. P. (1996). State bounding with ellipsoidal set description of the uncertainty. *International Journal of Control*, 65(5):847–866.

[Menage et al., 2014] Menage, O., Bethencourt, A., Rousseaux, P., and Prigent, S. (2014). Vaimos: Realization of an autonomous robotic sailboat. In Bars, F. L. and Jaulin, L., editors, *Robotic Sailing 2013*, pages 25–36. Springer International Publishing.

[Milanese et al., 1996] Milanese, M., Norton, J., Piet-Lahanier, H., and Walter, E., editors (1996). *Bounding Approaches to System Identification*. Plenum Press, New York, NY.

[Moore, 1966] Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ.

[Moore, 1979] Moore, R. E. (1979). *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA.

[Morel and Yu, 2009] Morel, J.-M. and Yu, G. (2009). Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2:438–44.

[Nedialkov, 2006] Nedialkov, N. (2006). Interval tools for odes and daes. In *Scientific Computing, Computer Arithmetic and Validated Numerics, 2006. SCAN 2006. 12th GAMM - IMACS International Symposium on*, pages 4–4.

[Nedialkov et al., 1999] Nedialkov, N., Jackson, K., and Corliss, G. (1999). Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68.

[Oshman and Davidson, 1999] Oshman, Y. and Davidson, P. (1999). Optimization of observer trajectories for bearings-only target localization. *Aerospace and Electronic Systems, IEEE Transactions on*, 35(3):892–902.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34.

[Röhrig and Müller, 2009] Röhrig, C. and Müller, M. (2009). Indoor location tracking in non-line-of-sight environments using a ieee 802.15.4a wireless network. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'09, pages 552–557, Piscataway, NJ, USA.

[Rusinkiewicz and Levoy, 2001] Rusinkiewicz, S. and Levoy, M. (2001). In proc. third int 3-d digital imaging and modeling conf. pages 145–155.

[Sliwka, 2011] Sliwka, J. (2011). *Using set membership methods for robust underwater localisation.* PhD dissertation, ENSTA Bretagne, Brest, France.

[Srikanth and Toueg, 1987] Srikanth, T. K. and Toueg, S. (1987). Optimal clock synchronization. *Journal of the Association for Computing Machinery*, 34(3):626–645.

[Szabo et al., 2006] Szabo, B., Szöllösi, G., Gönci, B., Jurányi, Z., Selmeczi, D., and Vicsek, T. (2006). Phase transition in the collective migration of tissue cells: experiment and model. *Physical Review E*, 74(6):061908.

[Szabó et al., 2009] Szabó, P., Nagy, M., and Vicsek, T. (2009). Transitions in a self-propelled-particles model with coupling of accelerations. *Physical Review E*, 79(2):021908.

[Thrun et al., 2005] Thrun, S., Bugard, W., and Fox, D. (2005). *Probabilistic Robotics.* MIT Press, Cambridge, M.A.

[Thrun et al., 2000] Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2000). Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141.

[Tu, 1999] Tu, X. (1999). *Artificial animals for computer animation: biomechanics, locomotion, perception, and behavior.* Number 1635. Springer.

[Tu and Terzopoulos, 1994] Tu, X. and Terzopoulos, D. (1994). Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50. ACM.

[Turgut et al., 2008] Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008). Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120.

[Veres et al., 2008] Veres, S., Tsourdos, A., and Fisher, M. (2008). Low cost disposable autonomous vehicles. *Defense Management Journal*, 42:64–65.

[Vicsek et al., 1995] Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., and Shochet, O. (1995). Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226.

[Vicsek and Zafeiris, 2012] Vicsek, T. and Zafeiris, A. (2012). Collective motion. *Physics Reports*, 517(3):71–140.

[Waltz, 1975] Waltz, D. (1975). Generating semantic descriptions from drawings of scenes with shadows. In Winston, P. H., editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, New York, NY.

[Yang and Medioni, 1992] Yang, C. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155.

[Zhang et al., 2009] Zhang, L.-c., Xu, D.-m., Liu, M.-y., and Yan, W.-s. (2009). Cooperative navigation and localization for multiple uuvs. *Journal of Marine Science and Application*, 8(3):216–221.

Résumé

Une bonne localisation est essentielle à toute créature terrestre souhaitant se déplacer dans son environnement. Les organes ou senseurs utilisés par les humains sont en partie les yeux et le système vestibulaire (oreille interne). Tout comme les humains, les robots ont besoin de capteurs pour se localiser, respectivement une caméra et un gyroscope. Cependant les machines n'ont pas encore les capacités du cerveau humain pour se localiser. C'est pourquoi il est important de développer des algorithmes efficaces de localisation, et plus particulièrement sous l'eau où il n'est pas possible d'utiliser le GPS. Les AUVs (Véhicules sous-marins autonomes) sont de plus en plus développés dans les domaines civils (recherche sismique, inspection d'installations marines, etc.) et militaires (guerre des mines). Je fus personnellement interviewé par *France 2* sur ce sujet qui s'étend de manière exponentielle (interview disponible sur http://www.aymericbethencourt.com/thesis/). L'avantage des AUVs comparés aux traditionnels ROVs (Véhicules télé-opérés) est leur habilité à se « débrouiller tout seul ». La mission est chargée dans l'AUV et la réalise sans intervention de l'opérateur. Une bonne localisation de l'AUV est alors essentielle.

A cela s'ajoute des problématiques de monté en échelle. Certains projets développés par la *Direction Général pour l'Armement*, ou de grandes compagnies telles que *CGG* ou *Saudi Aramco* prévoient la mise en place d'essaims de plusieurs milliers d'AUVs simultanément, et cela afin de réaliser des mesures sismiques. Les AUVs doivent alors se positionner en grille très précisément. L'utilisation de méthodes dites par intervalle semble donc justifiée dans ce contexte ou la localisation de l'AUV à besoin d'être garantie.

En effet, les méthodes par intervalle permettent de représenter des ensembles solution avec leur incertitude, garantissant qu'aucune solution n'existe en dehors de ses bornes. Nous verrons également que l'analyse par intervalles possède de nombreux outils particulièrement puissants pour notre problématique. Les méthodes par intervalles sont notamment très puissantes pour la résolution de larges systèmes d'équations, particulièrement adapté dans le cas de milliers d'AUVs.

Cette thèse est organisée comme suit. Le chapitre 1 introduit le sujet. Le chapitre 2 présente l'analyse par intervalle et son application à la localisation en robotique mobile. Le chapitre 3 étend la notion de « contracteur » et propose l'évaluation de contracteurs minimaux par transformations géométriques. Le chapitre 4 formalise le problème de localisation dynamique d'AUVs sous la forme d'un problème de satisfaction de contraintes inter-temporel, et propose un nouvel outil, le tube, pour résoudre ce problème. Le chapitre 5 s'intéresse ensuite à un modèle d'AUV plus proche de la réalité, pour lequel l'horloge du robot est désynchronisée par rapport au temps réel. Enfin le chapitre 6 conclu la thèse par une forte monté en échelle des algorithmes et simulations présentés dans les chapitres précédents.

Mots-clés : Localisation, intervalles, AUV, robotique, tube, contraintes inter-temporelles, mesures incertaines bornées.

Abstract

Localisation and spatial awareness are fundamental to any application involving mobility, and essential to life on earth. The human body needs to be localized in space to move towards a goal. Just like the human body, a good localisation is essential for any mobile robotics application. However robots do not (yet) have the same computing capability as the human brain. Robots often use the GPS, but what happens when it is not; or partially not, available? For many indoor or underwater applications, the GPS is simply not a solution as high frequency electromagnetic waves barely propagate in these environments.

As the number of underwater operations increases every year, the need for autonomous underwater robots becomes greater. There are many underwater robotics applications in the fields of oceanography, biology and wreck exploration. As this thesis has been realised under fundings of the *Délégation Général pour l'Armement* (DGA, the French Military Procurement Agency), military applications are also heavily considered, for which applications are area inspection and mine sweeping among others. In both the civilian and military fields, it is sometimes necessary to use a group or a swarm of robots that need to cooperate to realize a task. Projects like *SpiceRack* realized by *CGG* and *Saudi Aramco* will use a swarm of 3,000 underwater AUVs to realize seismic surveys of the seabed. For this, the AUVs in the swarm need to position themselves in a precise grid pattern, and stay in position despite potentially strong currents. It is important that the robots localize themselves effectively and quantify the error on their position. If the robot is not where "it thinks" it is, the seismic survey could be corrupted, or a robot could collide against another AUV.

In this context, the use of interval methods is relevant as they allow to represent the set of compatible solutions with their uncertainty. Moreover, interval analysis has several advantages. It works for non-linear equations without approximation, no solution can exist outside the bounds of the interval, and finally, interval methods are a very fast for solving systems with a high number of equations, which seems particularly adapted for a swarm of robots.

This thesis is organised as follows. Chapter 1 introduces the subject. Chapter 2 presents interval analysis and its application to mobile robotics. Chapter 3 extends the notion of "contractor" and proposes the evaluation of minimal contractors using geometrical transformations. Chapter 4 formalises the problem of dynamically localising a group of AUVs in the form of an inter-temporal constraints satisfaction problem, and introduces a new tool, the tube, to solve this problem. Chapter 5 then considers a more realistic AUV model, which clock is not synchronised with the real time. Finally the chapter 6 concludes the thesis with a strong increase of the number of AUVs to prove the scalability of the algorithms presented in the previous chapters.

Keywords: Localisation, intervals, AUV, robotics, tube, inter-temporal constraints, bounded uncertain measurements.