Acta Cybernetica **26** (2024) 871–887.

The Codac Library: A Catalog of Domains and Contractors

Simon Rohou[ab], Benoît Desrochers[ac], and Fabrice Le Bars[ad]

Abstract

Codac (Catalog Of Domains And Contractors) is a C++/Python library providing tools for constraint programming over reals, trajectories and sets. It has many applications in parameter estimation, guaranteed integration or robot localization and provides reliable outputs by computing sets of feasible solutions according to the constraints defining the problem. This paper provides a brief overview of the library and its Contractor Network approach, illustrated on a convincing robotic application.

Keywords: constraint programming, interval analysis, state estimation, dynamical systems, solver, robotics, Contractor Network, SLAM

1 Introduction

This paper provides an overview of the Codac library[1] (http://codac.io), that aims at providing a catalog of tools based on interval analysis and constraint programming. The toolbox allows to approximate feasible solutions of non-linear and/or differential systems. Since the solution of these complex systems cannot generally be calculated exactly, Codac uses numerical analysis to compute the bounds of sets of feasible solutions. The assets are *guarantee* (computations are guaranteed to never lose solutions, due to the rigorous interval arithmetic) and *exhaustiveness* (if multiple values are possible, all of them are characterized). In the same way, obtaining an empty set allows to safely disprove properties of a system. In Codac, the variables can be of different types, such as reals, vectors [2], trajectories [35], uncertain sets [9], graphs [18], *etc.*, in order to address a wide range of problems.

Intervals are used to reliably propagate uncertainties (from sensors, models, discretizations), even in the case of non-linearities, provided that they are bounded. Coupled with constraint programming, these methods have been shown to be effective for solving complex problems involving constraints that are generally difficult to

[a]ENSTA Bretagne, Lab-STICC, UMR CNRS 6285, Brest, France
[b]E-mail: simon.rohou@ensta-bretagne.fr, ORCID: 0000-0001-6232-9918
[c]E-mail: benoit.desrochers@ensta-bretagne.org, ORCID: 0000-0001-7910-1119
[d]E-mail: fabrice.le_bars@ensta-bretagne.fr, ORCID: 0000-0001-9413-4621
[1]Codac has been built upon two former libraries: Tubex and pyIbex.

DOI: 10.14232/actacyb.302772

handle (strong uncertainties, differential equations, temporal delays [39], indistinguishable data, inter-temporal measurements [36], hybrid systems, *etc.*). Classical robotic applications such as Simultaneous Localization And Mapping (SLAM), the kidnapped robot problem, or the exploration of unstructured environments, can be formalized as such systems, and are still challenging issues. Conventional methods such as Kalman or particle filters are commonly used for tackling these problems. However, they have limitations related to the context of use, the propagation of error distributions, the reliability of the outputs, or the involved equations describing the system. Besides, constraint programming coupled with interval methods allows to handle a wider class of systems and has been shown to be comfortable with solving several problems known to be difficult, in a very few steps with the simplicity offered by constraint programming [7].

Recent advances in interval methods have been done by the community, and the Codac library gathers a part of related state-of-the-art implementations with the objective to make them easy to combine. This paper provides an overall picture of the library and proposes an application on an academic problem of SLAM.

2 Constraint programming

Constraint programming is a paradigm in which users concentrate on the properties of a solution to be found (*e.g.* the pose of a robot, the location of a landmark, the orbit of a satellite) by stating constraints on variables. Constraints usually come from the equations of the problem, inequalities, or measurements from sensors. The variables appear in the equations. Then, a solver performs *constraint propagation* on the variables and reliably provides feasible solutions corresponding to the problem. In this approach, the user concentrates on *what* the problem is instead of *how* to solve it, thus leaving the computer to deal with the *how*. The strength of this declarative paradigm lies in its simplicity, as it allows one to describe a complex problem without requiring knowledge of resolution tools and their specific parameters to tune. The second asset is genericity: a situation is seen from a high-level point of view and this abstraction enables the resolution of a wide range of problems. The energy is mainly spent on the description of the problem.

Since several decades, the community of constraint programming has brought numerous contributions for dealing with discrete problems. The Prolog language [3] appears to be one of the most famous outcomes with free implementations available to the community. Classical applications of these developments lie in automated planning or interactive theorem proving. While a major effort from the community has been undertaken around this concept, other studies appeared in order to tackle continuous problems with this paradigm [17]. For both discrete and continuous problems, constraint programming can be applied by defining a Constraint Network involving variables $\mathcal{V}_i$, domains $\mathcal{D}_i$, and constraints $\mathcal{L}_j$ [25].

Variables In continuous problems, the members of a system, including the unknown solutions, are reals $x \in \mathbb{R}$ or vectors $\mathbf{x} \in \mathbb{R}^n$. Recent advances have led to

the extension of Constraint Networks in order to tackle a larger number of types of variables, such as sets $\mathbb{X} \in \mathcal{P}(\mathbb{R}^n)$ [11], graphs [18], paths [23], *etc.* In particular, dynamical systems can be drawn as Constraint Networks by introducing so-called trajectories variables, denoted by $\mathbf{x}(\cdot) : \mathbb{R} \to \mathbb{R}^n$, thus allowing to consider the solution of a dynamical system as a single and continuous item.

Domains A variable $\mathcal{V}_i$ is known to be enclosed in some domain $\mathcal{D}_i$ on which we will apply constraints $\mathcal{L}_j$ *via* some operators. Domains define non-empty ranges of feasible values. For instance, domains can be intervals, polytopes, ellipsoids, subpavings, tubes, *etc.*

Constraints Elementary facts and rules apply on variables: so-called constraints. There are very few restrictions on the forms of the constraints: they are understood as the expression of any relation that binds variables, which are known to belong to some domains. In the continuous and differential context, constraints may be non-linear equations such as $x_3 = \cos(x_1 + \exp(x_2))$, inequalities, quantified parameters [13], differential systems expressed as $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$, *etc.* For instance, in the context of robotics, constraints will come from state equations, numerical models, or measurements. Uncertainties from sensors are specified either by other constraints (*e.g.* $x \geqslant 0$) or by restricting the domains of the variables (*e.g.* $[x] = [0, \infty]$, where $[x]$ is the interval domain known to enclose the variable x).

Constraint propagation Elementary constraints are relations that cannot be decomposed, such as $c = a + b$. Then, complex constraints can be considered by combining simpler constraints, in order to increase in complexity, while preserving simplicity. This leads to a propagation process, due to dependencies between constraints sharing the same variables.

The aim of Codac is to easily deal with constraints in order to eventually characterize sets of values compliant with the defined rules. This is done by providing implementations of operators for elementary constraints as well as algorithms for their combination. The related domains and operators depend on tools from interval analysis.

3 Interval methods for constraint programming

In the constraint programming approach, the estimation of a variable consists in reducing its domain. The obtained set is said to be reliable: the resolution must guarantee that no solution will be lost during the solving process, according to the constraints defining the problem. In practice, domains and constraints have to be numerically computed. The use of interval methods is perfectly suited for this, providing intervals for domains and a rigorous arithmetic to implement constraints.

Intervals domains An interval $[x]$ is a closed and connected subset of $\mathbb{R}$. The set of all intervals is denoted $\mathbb{IR}$. An interval vector (a box) $[\mathbf{x}]$ of $\mathbb{IR}^n$ is an axis-aligned box, a closed and connected subset of $\mathbb{R}^n$. These interval sets can be easily represented in a computer. For instance, a box $[\mathbf{x}] = [\mathbf{x}^-, \mathbf{x}^+]$ will be defined by its two bounds $\mathbf{x}^-$ and $\mathbf{x}^+$, that are representable vectors of $\mathbb{R}^n$. Intervals can be extended to trajectories: we then define a *tube* $[\mathbf{x}](\cdot) : \mathbb{R} \to \mathbb{IR}^n$ as an interval of two trajectories $\mathbf{x}^-(\cdot)$ and $\mathbf{x}^+(\cdot)$ such that $[\mathbf{x}](\cdot) = [\mathbf{x}^-(\cdot), \mathbf{x}^+(\cdot)]$. They have also been extended to sets with *thicksets* denoted by $[\![\mathbb{X}]\!] = [\mathbb{X}^-, \mathbb{X}^+]$ where $\mathbb{X}^\pm \in \mathcal{P}(\mathbb{R}^n)$.

Contractors A contractor $\mathcal{C}$ for a constraint $\mathcal{L}$ is an operator designed to reduce a domain without losing any solution consistent with $\mathcal{L}$. A contractor is thus an algorithm that can act on an interval domain for narrowing its bounds in a reliable way. The following definition applies for contractors on boxes [7], and can be easily extended to tubes, thicksets, *etc.*

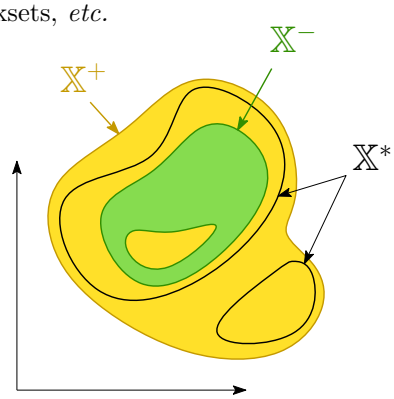


Figure 1: A contractor $\mathcal{C}_\mathbb{X}$ related to a constraint representing a set $\mathbb{X}$ is applied on several boxes. Hatched parts correspond to vectors that are removed after the contraction.

Definition. *A contractor $\mathcal{C}$, associated with a constraint $\mathcal{L}$, on a box $[\mathbf{x}] \in \mathbb{IR}^n$ is a mapping from $\mathbb{IR}^n$ to $\mathbb{IR}^n$ such that:*

$$(i) \quad \forall [\mathbf{x}] \in \mathbb{IR}^n,\ \mathcal{C}([\mathbf{x}]) \subseteq [\mathbf{x}], \qquad \textit{(contraction)}$$

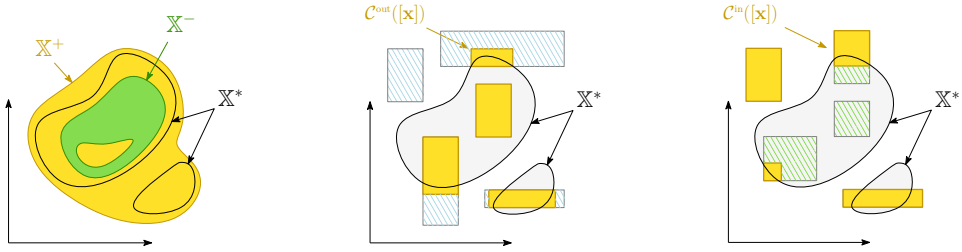
$$(ii) \quad \begin{pmatrix} \mathcal{L}(\mathbf{x}) \\ \mathbf{x} \in [\mathbf{x}] \end{pmatrix} \Longrightarrow \mathbf{x} \in \mathcal{C}([\mathbf{x}]). \quad \textit{(consistency)}$$

The use of contractors allows to enclose complex algorithms in simple black boxes that are only used to contract a domain according to a constraint, in a reliable way. The reliable property (expressed by the *consistency* rule of Definition 3) is important as it allows to combine contractors, call them in any order and as much as necessary, without running the risk of losing solutions. This allows to deal with complex problems providing that contractors are at hand and sufficient to address elementary constraints obtained from a decomposition. Finally, domains

and contractors can be combined in propagation algorithms in order to implement an interval solver.

One may emphasize that interval methods have the reputation of being limited to problems of low dimensions, due to usual bisections of the domains which leads to an exponential complexity order. Contractors may overcome this problem: they are usually given by polynomial-time algorithms and can be employed without performing bisections, which allows to tackle problems of higher dimensions.

Separators A contractor $\mathcal{C}^{\text{out}}$ associated with a set $\mathbb{X}$ aims at removing infeasible solutions (*i.e.* vectors that are not in $\mathbb{X}$) from a domain. When employed in a branch-and-contract algorithm, it allows to compute an outer approximation $\mathbb{X}^+ \supset \mathbb{X}$. However, the same contractor cannot be used for inner approximations $\mathbb{X}^-$, that are sets in which any vector is solution, that is $\mathbb{X}^- \subset \mathbb{X}$. For instance, when no contraction happens on a given box, *i.e.* $\mathcal{C}^{\text{out}}([\mathbf{x}]) = [\mathbf{x}]$, it is not possible to know if $[\mathbf{x}]$ is completely included in $\mathbb{X}$ (see the yellow box on Fig. 1) or if there may exist vectors in $[\mathbf{x}]$ that do not belong to $\mathbb{X}$ (see the red dot in Fig. 1). Inner approximations are particularly important for proving the existence of solutions. Their computation can be done by using a contractor $\mathcal{C}^{\text{in}}$ consistent with the complementary $\overline{\mathbb{X}}$ that removes vectors of $\mathbb{X}$, as depicted in Fig. 2. The pair $\{\mathcal{C}^{\text{in}}, \mathcal{C}^{\text{out}}\}$ defines a new operator called a *separator* [19], which can be employed in a branch-and-contract algorithm in order to characterize simultaneously an inner $\mathbb{X}^-$ and an outer approximation $\mathbb{X}^+$ and therefore enclose $\mathbb{X}$ in a *thickset* $[\![\mathbb{X}]\!] = [\mathbb{X}^-, \mathbb{X}^+]$.



(a) $\mathbb{X}^* \in [\mathbb{X}^-, \mathbb{X}^+]$. (b) Illustration of $\mathcal{C}^{\text{out}}$. (c) Illustration of $\mathcal{C}^{\text{in}}$.

Figure 2: Enclosure of a set $\mathbb{X}^* \subset \mathbb{R}^2$ by two approximated inner and outer sets $\mathbb{X}^-$ and $\mathbb{X}^+$, computable using a pair of contractors (a separator) involved in a branch-and-contract algorithm. Note that any vector of $\mathbb{X}^-$ also belongs to $\mathbb{X}^+$. For computing $\mathbb{X}^+$, resp. $\mathbb{X}^-$, a $\mathcal{C}^{\text{out}}$ contractor, resp. $\mathcal{C}^{\text{in}}$, can be employed as pictured in Subfig. 2b–2c. The green hatched areas are part of $\mathbb{X}^-$ while the blue ones belong to $\overline{\mathbb{X}^+}$. In practice, these algorithms output subpavings made of non-overlapping boxes, not represented in Subfig. 2a.

Interval domains, contractors and separators form a set of items that one can combine in order to describe continuous nonlinear equations, dynamical systems, or measurements, that are usually encountered in physics or robotics. They are the basic components of the Codac library.

4 The Codac library

4.1 A framework of domains and contractors

The API of Codac can be broken down into three layers: (*i*) a list of implemented domains such as intervals, boxes, tubes, thicksets, *etc.*; (*ii*) a catalog of contractors and separators for dealing with a wide variety of constraints; (*iii*) a top-level system solver called Contractor Network. Recent efforts from the community [30, 39, 19] have led to new domains, contractors and separators. The objective of Codac is to gather the related implementations and form a catalog of algorithms associated with publications from the literature.

4.2 Other libraries

Several libraries, see for instance filib++ [27], MPFI [32] and GAOL [14], provide low-level features related to interval arithmetic, most of them including reliable numerical operations with outward rounding. For its interval arithmetic computations, Codac currently stands on GAOL that has shown good performances and a large portability on operating systems. At a higher level of abstraction, the contractor programming approach [7] deployed in Codac is also a cornerstone of the IBEX library [8]. Codac is, however, not limited to constraint processing over reals and is inspired by robotic applications, including the capacity to deal with dynamical systems, inter-temporal constraints and thicksets.

The guaranteed simulation of dynamical systems is also the topic of several set-based libraries, namely Vnode [26], Cosy [31], DynIBEX [1] or CAPD [41]. These libraries have applications in robotics and automatic control, by verifying dynamical properties of non-linear systems [28] or for the computation of reachable sets [15]. They are also used by mathematicians to prove conjectures [12]. While they offer good performances for guaranteed integrations, they are mainly suited for systems expressed under the form of Initial Value Problems (IVPs), that is $\{\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}),\ \mathbf{x}(0) \in [\mathbf{x}_0]\}$, which does not represent the diversity of state estimation problems. This limitation motivated new constraint-based approaches in order to assess for instance inter-temporal relations, time uncertainties, delays, or hybrid systems. Robotics also requires to mix various uncertainties, not only related to dynamical systems, but also involving sets, graphs, paths, to name but a few. Nonetheless, future work may focus on building bridges between these libraries in order to benefit from good performances from each specific tool. The contractor framework could ease the use of these state-of-the-art contributions by enclosing them in contractor operators compatible with each other.

Finally, the above-mentioned libraries are mainly available in C++. The core of Codac is also developed in C++ for performance and historical reasons, and in order to be compliant with many robotic frameworks. In addition, a Python wrapper of the library allows to use Codac in Python 3 while benefiting from C++ performance. The library is actively supported on Ubuntu derivatives, Windows and macOS, with pre-built packages available.

4.3 Implemented domains

The building blocks of this library are intervals and the above-mentioned domains (boxes, tubes, thicksets, *etc.*) are mainly designed as compositions of intervals. However, for specific computation needs and in order to avoid as much as necessary wrapping effects, that are pessimism drawbacks induced by the use of inaccurate enclosures, some implementations in Codac rely on more specific domains, such as cuboids [24], polytopes [40] or ellipsoids [21, 29]. For instance, it has been shown in [33] that the integration of continuous-time linear systems could be computed exactly without pessimism by using polytopes instead of boxes, at the expense of longer computation times. A good trade-off has been studied in [30] with the use of ellipsoids for enclosing the continuous states. These are specific domains also available in Codac, and relevant for guaranteed integration purposes.

Domains for trajectories: tubes $\mathbb{X}(\cdot)$ In Codac, a tube is implemented as a temporal sequence of sets $\mathbb{X}_i$, where the $\mathbb{X}_i$s can be intervals, boxes, ellipsoids [30], polytopes, subpavings, or any domain[2] for which set operations can be computed. More precisely, a tube $\mathbb{X}(\cdot)$ with a sampling time $\delta > 0$ is implemented as a set-valued function which is constant for all t inside intervals $[k\delta, k\delta + \delta]$, $k \in \mathbb{N}$. The set $[k\delta, k\delta + \delta] \times \mathbb{X}(t_k)$, with $t_k \in [k\delta, k\delta + \delta]$, is called the k^{th} *slice* of the tube $\mathbb{X}(\cdot)$ and is denoted by $\mathbb{X}(k)$. For instance, when the $\mathbb{X}_i$s correspond to intervals, the implementation amounts to an interval of step functions $[\underline{x^-}(\cdot), \overline{x^+}(\cdot)]$ such that $\forall t,\ \underline{x^-}(t) \leqslant \overline{x^+}(t)$, as pictured by Fig. 3.

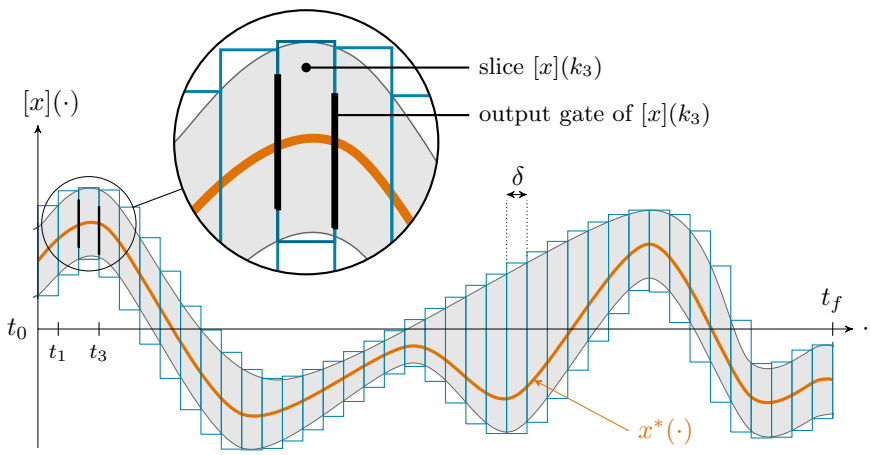


Figure 3: An interval-tube $[x](\cdot)$ implemented as a list of interval-slices. In practice, the sampling δ is not necessarily constant.

[2]Generic programming is enabled for tubes thanks to their C++ implementation providing compile-time templates.

This implementation then takes rigorously into account floating point precision when building a tube, thanks to reliable numerical libraries. Further computations involving $\mathbb{X}(\cdot)$ will be based on its slices, thus giving an outer approximation of the actual solution set. For instance, the lower bound of the integral of a boxed-tube $[\mathbf{x}](\cdot)$ is simply computed as the signed area of the region in the tx-plane that is bounded by the graph of $\underline{\mathbf{x}}^-(t)$ and the t-axis. The lower the slice width δ, the higher the precision of the approximation. Note that the current implementation allows a variable sampling, as well as input and output *gates* on each slice as pictured in Fig. 3 in the case of interval slices. In the literature, these gates are classical restrictions used when dealing with IVPs.

Other representations of tubes are available in Codac, and provide faster evaluations than with a simple sequence of slices. For instance, the computer evaluation $\big[\mathbb{X}([t])\big] = \big[\,\{x(t) \mid x(\cdot) \in \mathbb{X}(\cdot), t \in [t]\}\,\big]$, with $[t]$ a large interval over several slices, can be optimized with a redundant data structure such as binary trees [36, pp. 54–55] or polynomial approximations of the bounds of the tubes. This is relevant when many evaluations have to be performed on tubes that have non-updated values, namely integral calculations or tube inversions [35].

Domains for sets: thicksets $[\![\mathbb{X}]\!]$ A set $\mathbb{X} \subset \mathbb{R}^n$ can be bracketed between two sets $\mathbb{X}^-$ and $\mathbb{X}^+$, forming a so-called thickset $[\![\mathbb{X}]\!] = [\mathbb{X}^-, \mathbb{X}^+]$, [11]. The approximation of $\mathbb{X}^-$ and $\mathbb{X}^+$ is possible using subpavings [20] that are unions of non-overlapping boxes of $\mathbb{R}^n$. As for tubes, optimized implementations based on binary trees allow to speed up evaluations of subpavings. Fig. 4 illustrates an application of Codac involving tubes and thicksets for the computation of the guaranteed zone explored by a robot [9]. This example typically illustrates the need to couple various domains for robotic applications.

4.4 Contractors and separators

A list of contractors (or separators for approximating sets) is available in the library. They aim at contracting domains in a reliable way and do not need to be configured. Most of them implement elementary constraints. Other contractors are compositions of primitive contractors based on syntax trees, see for instance the HC4revise contractor [2] available in IBEX and Codac. This allows one to obtain a high-level contractor built from an analytical expression, and automatically involving primitive contractors. Besides, avoiding constraint decompositions can provide better results if one relies on a dedicated and optimized contractor. For instance, in the case of the polar equation ($x = \rho\cos(\theta)$, $y = \rho\sin(\theta)$), the contractor $\mathcal{C}_{\text{polar}}([x],[y],[\rho],[\theta])$ provided by [10] allows a minimal contraction for $[x]$, $[y]$, $[\rho]$ and $[\theta]$. Additional contractors are designed to deal with inter-temporal and differential constraints, such as linear systems $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ ([33]), temporal delays $x(t) = y(t-a)$ ([39]), time uncertainties $\{y = x(t), t \in [t]\}$ ([35]), differential nonlinear equations $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ ([5, Chap. 4]), to name but a few. Other contractors focus on geometric constraints [16], or allow contractions robust to outliers [6].

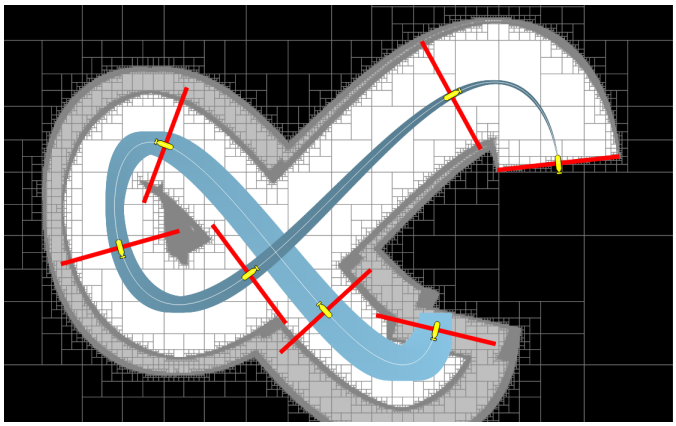


Figure 4: Guaranteed zone explored by a robot [9]. A robot equipped with a side-scan-sonar (the scope of which is pictured by red lines) evolves along a trajectory $\mathbf{x}^*(\cdot)$ (white line) estimated with uncertainties (blue tube $[\mathbf{x}](\cdot)$). The problem consists in computing the explored space $\mathbb{X}$. Taking into account the increasing uncertainty of localization, $\mathbb{X}$ cannot be computed exactly. However, an inner set $\mathbb{X}^-$ (in white) can be approximated, and corresponds to the part that has been surely observed with the sonar. Also, the black set (*i.e.* $\overline{\mathbb{X}^+}$) is computable and related to parts of the environment for which we can reliably state that no observation has been done, for any feasible trajectory $\mathbf{x}(\cdot) \in [\mathbf{x}](\cdot)$. The gray part corresponds to the *penumbra* in which observations may have been done, or not.

5 Contractor Networks (CN)

5.1 Higher degree of abstraction

When several contractors are at stake, there may be interactions between them: a contraction from one contractor may trigger another one, which reveals a constraint propagation process [4, 7]. It becomes necessary to call some contractors several times in order to converge to the best contraction of the domains. This number of contracting iterations cannot be known in advance. It depends on the contractors at stake, their efficiency and their sequencing. Classically, one can implement a loop of contractions in order to process the contractors as long as there is a contraction on one of the domains. The iteration stops when a fixed point has been reached: when nothing can be contracted anymore. Note that because a computer uses floating point numbers, the iterative fixed point will be necessarily reached in a finite number of steps [22, p. 42]. In practice, we may stop the iteration as soon as the contractions are not significant anymore. In any case, even if the algorithm stops before reaching the fixed point, the actual solution will always be enclosed in the domains.

Codac provides a high-level propagation tool, called a Contractor Network (CN),

that aims at managing the propagation process automatically. This simplifies the use of contractors: the user does not have to implement contracting loops and to manage stopping conditions. Instead, he/she only has to design a CN by connecting contractors and domains together. This approach allows to take a higher degree of abstraction by hiding the propagation part. What remain are the domains and contractors, which correspond to the variables and their related rules, as in a pure declarative paradigm.

5.2 Refined propagation process

In addition, the propagation process can be enhanced: some heuristic encoded in the CNs can allow a better sequence of the contractors calls. While the user only states the relations between contractors and domains, the CN defines by itself the sequence of contractions to be run, depending of the types of domains and contractors and some empirical models encoded in the library. The result is a global contraction that runs faster than with a random sequence of contractors.

Numerically, a contractor on a complex domain (such as a tube, a thickset) may amount to several contractors on subdomains (such as a slice, a box). For instance, the $\mathcal{C}_+$ contractor for the constraint $a = b + c$ can be extended to tubes: $\mathcal{C}_+\big([x_1](\cdot), [x_2](\cdot), [x_3](\cdot)\big)$, [35]. This does not correspond to inter-temporal or differential equations: the related constraint applies for all t. This amounts to calling $\mathcal{C}_+$ for all tuples of slices $\big([x_1](k), [x_2](k), [x_3](k)\big)$. In practice, some parts of the tube may not be updated during a propagation and it becomes relevant to avoid further calls of contractors on the involved slices, if we can state that they will be ineffective. While it would be cumbersome to tune a propagation algorithm at this level of granularity, it becomes worthwhile to rely on an automatic tool that will call the contractors only on relevant parts of the complex domains. Hence, CNs are also used to break tubes down into graphs of slices, each of them being connected to the former contractors related to the tube itself. The propagation algorithm then naturally calls the contractors if necessary. It leads to faster computations. To our knowledge, this is the first time that contractors are involved in a propagation network with heterogeneous domains such as tubes or thicksets.

5.3 Graphs of contractors and domains

A Contractor Network is a graph of domains and contractors. Fig. 5 provides an illustration of such graph, with domains pictured by circles and contractors drawn by boxes. A possible propagation sequence is the following: assume that $\mathcal{C}_3$ is first triggered, either manually or because the contractor has been newly added in the graph. $\mathcal{C}_3$ is added in a stack of contractors $\mathscr{L}$ that was empty so far. Then $\mathcal{C}_3$ is called as it is the first (and only) item in the stack. This results in a contraction of $[\mathbf{b}]$, which induces the addition of the connected contractors $\mathcal{C}_2$, $\mathcal{C}_4$, $\mathcal{C}_3$ in $\mathscr{L}$. $\mathcal{C}_2$ is then called (first item in the stack), which contracts again $[\mathbf{b}]$: $\mathcal{C}_2$ is added again in $\mathscr{L}$. $\mathcal{C}_4$ is now called and contracts $[\mathbf{a}]$, which adds $\mathcal{C}_1$, $\mathcal{C}_4$ in the stack. This sequence runs until $\mathscr{L}$ becomes empty; a fixed point has been reached. One can

note that the graph may also be made of directed arcs, depending on the involved constraints. This enhances the propagation and so computation times.

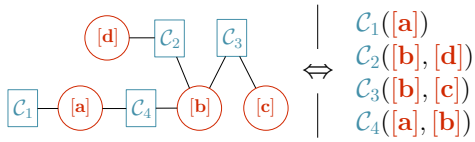


Figure 5: A CN corresponding to four contractors and four domains.

6 Example of use: a SLAM problem

6.1 Formalism

In robotics, Simultaneous Localization And Mapping (SLAM) [37] is a topic that ties together the problem of state estimation and that of mapping an unknown environment. A robot exploring its surroundings will associate localization uncertainties to the observed features of the environment, assigning their location with some error. However, a scene of the environment may be seen several times during the exploration, thus leading to an inter-temporal measurement which could benefit both localization and mapping procedures. Indeed, a robot that recognizes a part of the environment will deduce to be close to a previous position. This chicken-and-egg problem is difficult to solve with recursive algorithms such as Kalman filters, due to inter-temporal relations between the states, corresponding to so-called loop closures [38]. Because SLAM requires a capacity to manage equations involving states from different times and strong uncertainties, it can be more easily dealt with tubes and contractors.

We propose to solve a classical range-only SLAM problem using CNs. Let us consider a robot measuring distances from landmarks for which the position is unknown. This can be formalized with the following state equations:

$$\begin{cases} \mathbf{x}(0) = \mathbf{0}, & \text{(1a)} \\ \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), & \text{(1b)} \\ y^{(i)} = g(\mathbf{x}(t_i), \mathbf{b}^{(j)}), & \text{(1c)} \end{cases}$$

with $\mathbf{x}(t) \in \mathbb{R}^n$ and $\mathbf{u}(t) \in \mathbb{R}^m$, the state and input vectors, $y^{(i)} \in \mathbb{R}$ a distance measurement and $\mathbf{b}^{(j)} \in \mathbb{R}^2$ the related landmark of the environment. $\mathbf{f}$ and g are nonlinear functions depicting the evolution of the states and distance measurements. Both $\mathbf{u}(t)$ and $y^{(i)}$ are measured with some bounded errors.

6.2 Building a SLAM-CN

In a few steps, the problem is solved with Codac by (i) defining the initial domains (boxes, tubes) of our variables (vectors, trajectories); (ii) taking contractors from a

catalog of already existing operators, provided in the library, or building contractors for specific constraints; (*iii*) binding the contractors and domains in a CN; (*iv*) letting the CN solve the problem by contracting the sets of feasible values.

Eq. (1b) is a differential equation difficult to solve in the nonlinear case. We will therefore decompose the equation into two constraints $\mathbf{v} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and $\dot{\mathbf{x}} = \mathbf{v}$ in order to use available contractors, respectively $\mathcal{C}_{\mathbf{f}}$ [2] and $\mathcal{C}_{\frac{d}{dt}}$ [34]. Involved domains are intervals $[y]^{(i)}$, boxes $[\mathbf{b}]^{(j)}$ and tubes $[\mathbf{x}](\cdot)$, $[\mathbf{u}](\cdot)$, $[\mathbf{v}](\cdot)$. The resulting CN is pictured in Fig. 7, providing an illustration of the decomposition of tubes into graphs of slices for allowing a finer propagation. The source code of this SLAM-CN is available on `http://codac.io/slam` for encouraging future comparisons. The SLAM-CN runs contractions in less than 1s, providing the results pictured by Fig. 6. In this example, inter-temporal relations between the states are implicitly managed by the SLAM-CN. The landmarks are first localized and then used to improve the localization of the robot. This scenario is automatically managed by the CN.

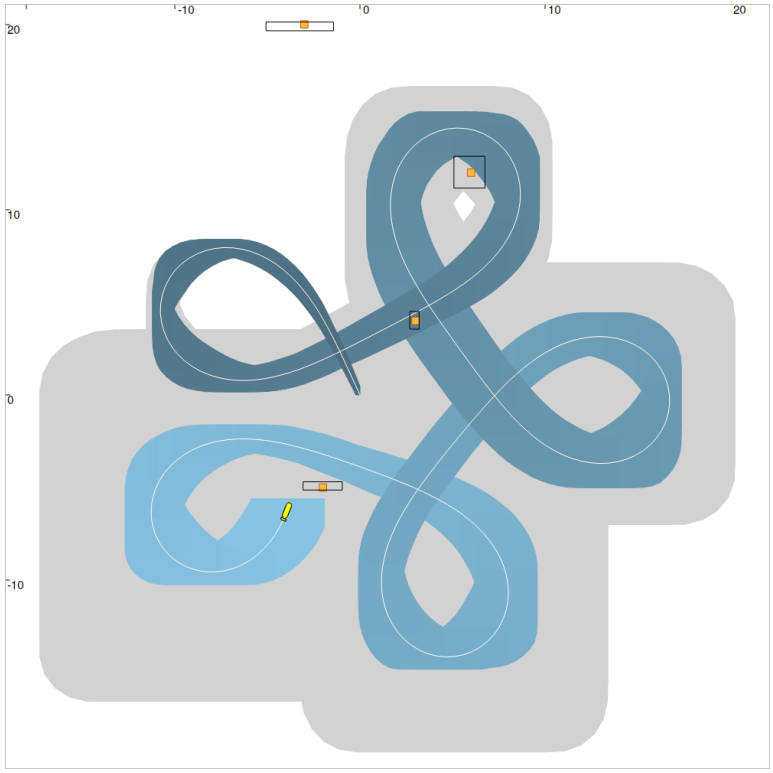


Figure 6: Contracted tube of positions (in blue) resulting from the SLAM-CN. The gray tube provides a reference corresponding to the localization drift without measurements. The estimated position of the landmarks is depicted by black boxes.

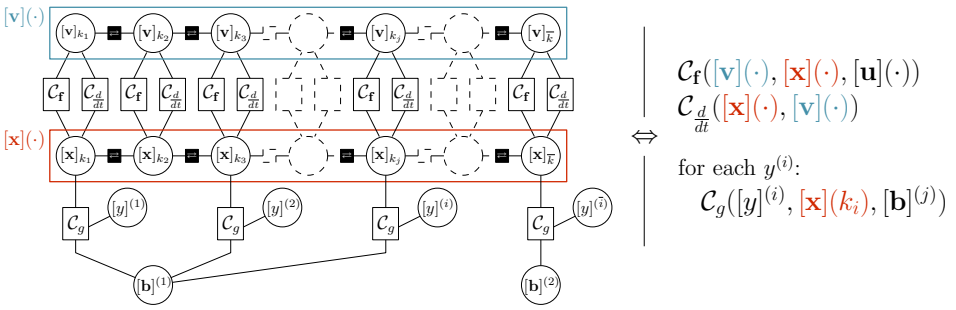


Figure 7: Illustration of a SLAM-CN associated with System (1) and some measurements from two landmarks $\mathbf{b}^{(1)}$, $\mathbf{b}^{(2)}$. The contractors and domains, formalized in a declarative way on the right-hand side, are transformed into a CN partially pictured on the left-hand side. For ease of reading, $[\mathbf{u}](\cdot)$ is not represented on this figure. The graph reveals how inter-temporal relations are implicitly built.

7 Conclusion

The Codac library offers a catalog of domains and contractors that one can combine into Contractor Networks in order to build interval solvers using a declarative paradigm. The assets lie both in the simplicity of the approach and the reliability of the results. It also allows to deal with a wide class of constraints that are classically encountered in real applications. Future work will concentrate on the development of the catalog of domains, contractors and separators, as well as improvements of CNs for real-time implementations.

The library is released under LGPLv3. The list of contributors of Codac is available on the website of the library, with related publications associated with the provided tools: http://codac.io. We encourage anyone interested in contributing to this open-source project to contact us.

References

[1] Alexandre dit Sandretto, J. and Chapoutot, A. Validated explicit and implicit Runge-Kutta methods. *Reliable Computing*, 22, 2016. URL: https://hal.archives-ouvertes.fr/hal-01243053.

[2] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J. F. Revising hull and box consistency. In *Proceedings of the International Conference on Logic Programming*, pages 230–244, Las Cruces, NM, 1999. DOI: 10.5555/341176.341208.

[3] Benhamou, F. and Touraivane, T. Prolog IV : langage et algorithmes. In *JFPLC*, pages 51–64, 1995. URL: https://www.mcours.net/cours/pdf/leilclic2/leilclic294.pdf.

[4] Bessiere, C. Constraint Propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Foundations of Artificial Intelligence*, Volume 2 of *Handbook of Constraint Programming*, pages 29–83. Elsevier, 2006. DOI: 10.1016/S1574-6526(06)80007-6.

[5] Bourgois, A. *Safe and collaborative autonomous underwater docking*. PhD dissertation, ENSTA Bretagne, Brest, France, 2021. URL: https://hal.science/tel-03151588v1.

[6] Carbonnel, C., Trombettoni, G., Vismara, P., and Chabert, G. Q-intersection algorithms for constraint-based robust parameter estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2630–2636. AAAI Press, 2014. DOI: 10.1609/aaai.v28i1.9117.

[7] Chabert, G. and Jaulin, L. Contractor programming. *Artificial Intelligence*, 173:1079–1100, 2009. DOI: 10.1016/j.artint.2009.03.002.

[8] Chabert, G., Ninin, J., and Neveu, B. IBEX: A C++ numerical library based on interval arithmetic and constraint programming. URL: http://www.ibex-lib.org.

[9] Desrochers, B. and Jaulin, L. Computing a guaranteed approximation the zone explored by a robot. *IEEE Transaction on Automatic Control*, 62:425–430, 2016. DOI: 10.1109/TAC.2016.2530719.

[10] Desrochers, B. and Jaulin, L. A minimal contractor for the polar equation; application to robot localization. *Engineering Applications of Artificial Intelligence*, pages 83–92, 2016. DOI: 10.1016/j.engappai.2016.06.005.

[11] Desrochers, B. and Jaulin, L. Thick set inversion. *Artificial Intelligence*, 249:1–18, 2017. DOI: 10.1016/j.artint.2017.04.004.

[12] Goldsztejn, A., Hayes, W., and Collins, P. Tinkerbell is chaotic. *SIAM Journal on Applied Dynamical Systems*, 10(4):1480–1501, 2011. DOI: 10.1137/100819011.

[13] Goldsztejn, A. and Jaulin, L. Inner and outer approximations of existentially quantified equality constraints. In *Principles and Practice of Constraint Programming*, pages 198–212, Nantes (France), 2006. DOI: 10.1007/11889205_16.

[14] Goualard, F. GAOL: Not just another interval library. URL: https://github.com/goualard-f/GAOL.

[15] Goubault, E., Mullier, O., Putot, S., and Kieffer, M. Inner approximated reachability analysis. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, pages 163–172, New York, NY, USA, 2014. ACM. DOI: 10.1145/2562059.2562113.

[16] Guyonneau, R. *Set-membership approaches for mobile robot localization*. PhD dissertation, Université de Nantes Angers Le Mans, France, 2013. URL: https://theses.hal.science/tel-00961501.

[17] Hansen, E. R. and Sengupta, S. Global constrained optimization using interval analysis. In Nickel, K., editor, *Interval Mathematics*, pages 25–47. Academic Press, New York, NY, 1980.

[18] Jaulin, L. Range-only SLAM with indistinguishable landmarks; a constraint programming approach. *Constraints*, 21, 2016. DOI: 10.1007/s10601-015-9231-9.

[19] Jaulin, L. and Desrochers, B. Introduction to the algebra of separators with application to path planning. *Engineering Applications of Artificial Intelligence*, 33:141–147, 2014. DOI: 10.1016/j.engappai.2014.04.010.

[20] Jaulin, L. and Walter, E. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993. DOI: 10.1016/0005-1098(93)90106-4.

[21] Kurzhanski, A. and Valyi, I. *Ellipsoidal calculus for estimation and control*. Birkhäuser, Boston, MA, 1997. URL: https://link.springer.com/book/9780817636999.

[22] Le Mézo, T. *Bracketing largest invariant sets of dynamical systems*. PhD dissertation, Université Bretagne Loire, Brest, France, 2019.

[23] Le Mézo, T., Jaulin, L., and Zerr, B. Bracketing the solutions of an ordinary differential equation with uncertain initial conditions. *Applied Mathematics and Computation*, 318:70–79, 2018. DOI: 10.1016/j.amc.2017.07.036.

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

[25] Mackworth, A. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. DOI: 10.1016/0004-3702(77)90007-8.

[26] Nedialkov, N. and Jackson, K. ODE Software that computes guaranteed bounds on the solution. In *Advances in Software Tools for Scientific Computing*, Lecture Notes in Computational Science and Engineering, pages 197–224. Springer, Berlin, Heidelberg, 2000. DOI: 10.1007/978-3-642-57172-5_6.

[27] Nehmeier, M. and von Gudenberg, J. W. filib++ , expression templates and the coming interval standard. *Reliable Computing*, 15:312–320, 2011. URL: https://interval.louisiana.edu/reliable-computing-journal/volume-15/no-4/reliable-computing-15-pp-312-320.pdf.

[28] Ramdani, N. and Nedialkov, N. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162, 2011. DOI: 10.1016/j.nahs.2010.05.010.

[29] Rauh, A. and Jaulin, L. A computationally inexpensive algorithm for determining outer and inner enclosures of nonlinear mappings of ellipsoidal domains. *Applied Mathematics and Computer Science*, 31(3):399–415, 2021. DOI: 10.34768/amcs-2021-0027.

[30] Rauh, A., Rohou, S., and Jaulin, L. An ellipsoidal predictor-corrector state estimation scheme for linear continuous-time systems with bounded parameters and bounded measurement errors. *Frontiers in Control Engineering*, 3, 2022. DOI: 10.3389/fcteg.2022.785795.

[31] Revol, N., Makino, K., and Berz, M. Taylor models and floating-point arithmetic: Proof that arithmetic operations are validated in COSY. *The Journal of Logic and Algebraic Programming*, 64(1):135–154, 2005. DOI: 10.1016/j.jlap.2004.07.008.

[32] Revol, N. and Rouillier, F. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005. DOI: 10.1007/s11155-005-6891-y.

[33] Rohou, S. and Jaulin, L. Exact bounded-error continuous-time linear state estimator. *Systems & Control Letters*, 153, 2021. DOI: 10.1016/j.sysconle.2021.104951.

[34] Rohou, S., Jaulin, L., Mihaylova, L., Le Bars, F., and Veres, S. M. Guaranteed computation of robot trajectories. *Robotics and Autonomous Systems*, 93:76–84, 2017. DOI: 10.1016/j.robot.2017.03.020.

[35] Rohou, S., Jaulin, L., Mihaylova, L., Le Bars, F., and Veres, S. M. Reliable non-linear state estimation involving time uncertainties. *Automatica*, 93:379–388, 2018. DOI: 10.1016/j.automatica.2018.03.074.

[36] Rohou, S., Jaulin, L., Mihaylova, L., Le Bars, F., and Veres, S. M. *Reliable robot localization: A constraint-programming approach over dynamical systems*. ISTE Ltd, London, 2019. DOI: 10.1002/9781119680970.

[37] Smith, R., Self, M., and Cheeseman, P. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer, New York, NY, 1990. DOI: 10.1007/978-1-4613-8997-2_14.

[38] Thrun, S. and Leonard, J. J. Simultaneous Localization and Mapping. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 871–889. Springer, Berlin, Heidelberg, 2008. DOI: 10.1007/978-3-540-30301-5_38.

[39] Voges, R. *Bounded-error visual-LiDAR odometry on mobile robots under consideration of spatiotemporal uncertainties*. PhD dissertation, Leibniz Universität Hannover, Germany, 2020. DOI: 10.15488/9932.

[40] Walter, E. and Piet-Lahanier, H. Exact recursive polyhedral description of the feasible parameter set for bounded-error models. *IEEE Transactions on Automatic Control*, 34(8):911–915, 1989. DOI: 10.1109/9.29443.

[41] Wilczak, D., Zgliczyński, P., Pilarczyk, P., Mrozek, M., Kapela, T., Galias, Z., Cyranka, J., and Capinski, M. Computer assisted proofs in dynamics group, a C++ package for rigorous numerics, 2017. URL: http://capd.ii.uj.edu.pl.