A boundary approach for set inversion

Luc Jaulin

A boundary approach for set inversion

Luc Jaulin

ENSTA-Bretagne, Lab-STICC, 2 rue F. Verny, 29200 Brest
lucjaulin@gmail.com

***Abstract*—In the paper, we present a new interval-based set inversion algorithm which takes into account the continuity of the problem. In the case where the set $\mathbb{Y}$ to be inverted has some volume, we show that inverting the boundary $\partial\mathbb{Y}$ of $\mathbb{Y}$ is sufficient to reconstruct the preimage $\mathbb{X} = \mathbf{f}^{-1}(\mathbb{Y})$. The inversion of $\partial\mathbb{Y}$ separates the domain of $\mathbf{f}$ into two regions : one inside $\mathbb{X}$ and one outside. To detect which part is inside or outside, we show that we can retro-propagate the information coming from $\mathbb{Y}$ at negligible cost. The efficiency of the approach is illustrated on a localization problem.**

***Index Terms*—Interval analysis, Constraint propagation, Continuous domains, Set Inversion**

I. Introduction

Notation. *In this paper, a vector* $\mathbf{x}$ *of* $\mathbb{R}^n$ *and a vector-valued function* $\mathbf{f}$ *will be written in bold font. An interval* $[x]$ *or an axis aligned box* $[\mathbf{x}]$ *will be written between brackets. The image by* $\mathbf{f}$ *of a box* $[\mathbf{x}]$ *will be written as* $\mathbf{f}([\mathbf{x}]) = \{\mathbf{y} | \exists \mathbf{x} \in [\mathbf{x}], \mathbf{y} = \mathbf{f}(\mathbf{x})\}$.

Given a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ and a set $\mathbb{Y} \subset \mathbb{R}^n$. Set inversion is the problem of characterizing the set $\mathbb{X}$ of all $\mathbf{x}$ inside a prior domain $\mathbb{X}(0) \subset \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{x}) \in \mathbb{Y}$. We have

$$\mathbb{X} = \mathbb{X}(0) \cap \mathbf{f}^{-1}(\mathbb{Y}) = \{\mathbf{x} \in \mathbb{X}(0) \,|\, \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\}. \quad (1)$$

Note that the function $\mathbf{f}$ is not necessarily invertible. For instance if $f(x) = x^2$, $\mathbb{Y} = [4, 16]$ and $\mathbb{X}(0) = [-5, 20]$ then

$$\begin{array}{rcl} \mathbb{X} & = & [-5, 20] \cap f^{-1}([4, 16]) \\ & = & [-3, 20] \cap ([-4, -2] \cup [2, 4]) \\ & = & [-3, -2] \cup [2, 4] \end{array} \quad (2)$$

Moreover, the solution set maybe empty. For instance, if $\mathbb{X}(0) = [-1, 1]$ instead of $[-5, 20]$ then

$$\mathbb{X} = [-1, 1] \cap f^{-1}([4, 16]) = \emptyset. \quad (3)$$

The set membership formalism is thus well suited to ill-posed problems involving uncertainties.

It is used in many engineering domains such as localization [8, 11], parameter estimation [17] [19], control [27], calibration [10], robotics [24], etc. Most guaranteed algorithms for set inversion are based on interval constraint propagation methods [25] [26]. They use a forward-backward contractor [1] [7] such as the HC4-revise algorithm [3]. Now, contractors have mainly be developed for solving equations [6] [23] or global optimization [14] where the solution set is reduced to some isolate points or surfaces and has no interior.

For set inversion, the solution set has an interior and exterior. Existing contractors are not able to retro-propagate properly the information coming from $\mathbb{Y}$. This leads to a poor outer approximation and unnecessary computations. It is sometimes possible to use the complementary set $\overline{\mathbb{Y}}$ to find an inner approximation [5], but, a part of the work done for contracting with respect to $\mathbb{Y}$ has to be repeated for $\overline{\mathbb{Y}}$, which should be avoided.

In this paper, we propose to inverse the boundary $\partial\mathbb{Y}$ of $\mathbb{Y}$ instead of inverting $\mathbb{Y}$. Due to the fact that $\partial\mathbb{Y} \subset \mathbb{Y}$, the corresponding contractor will more efficient. This inversion will provide an outer approximation for the boundary $\partial\mathbb{X}$ of the solution set $\mathbb{X}$. For the parts of the search space which are not in the boundary approximation, we need to show either they are inside or outside $\mathbb{X}$. We show that this information can be obtained with almost no extra computation cost by retro-propagating some binary information from $\mathbb{Y}$. To our knowledge, this is new in the community of constraint propagation.

The paper is organized as follows. Section II recalls the principle of set inversion algorithms and motivates the need of a more powerful propagation and the use of the boundaries. Section III presents in a new way, inspired from control theory, to present the forward-backward propagation. Section IV gives the boundary approach for set inversion which corresponds to the main contribution of this paper. Section V proposes an application to robot localization based on a *Time Difference Of Arrival* (TDOA). Section VI concludes the paper.

II. Motivation

Figure 1 illustrates the behavior of a typical contractor-based algorithm [16] to characterize a set $\mathbb{X}$ (magenta). The algorithm uses two types of contractors: a contractor $\mathcal{C}_{\mathbb{X}}$ for $\mathbb{X}$ and a contractor $\mathcal{C}_{\overline{\mathbb{X}}}$ for its complementary $\overline{\mathbb{X}}$. To characterize the part of $\mathbb{X}$ which is inside a box $[\mathbf{x}_0]$ the principle of the algorithm is the following

1) We have a set of boxes (green in the picture) which have to be studied. These green boxes are qualified as the *new boxes*. They have been created by a bisection process. At the initialization, we have a single new box: $[\mathbf{x}_0]$.
2) We contract each new box $[\mathbf{x}]$ using the contractors $\mathcal{C}_{\overline{\mathbb{X}}}$ and $\mathcal{C}_{\mathbb{X}}$. We get a box qualified as *contracted* (yellow in the picture).
3) A contracted box which is judged as too small it not bisected.

A zone that has been eliminated (or won) by a contractor is called a win-zone. The magenta part at the right of $[\mathbf{a}]$ was won by $\mathcal{C}_{\overline{\mathbb{X}}}$ and the blue part at the left of $[\mathbf{a}]$ was won by $\mathcal{C}_{\mathbb{X}}$.

We may observe the following properties:

1) The new boxes (green) are always obtained from a bisection of a contracted box, except $[\mathbf{x}_0]$. For instance boxes $[\mathbf{b}], [\mathbf{c}]$ are obtained from $[\mathbf{a}]$.

2) Contracted boxes (yellow) are always obtained from the contraction of a green box. For instance $[\mathbf{d}] = \mathcal{C}_{\mathbb{X}}([\mathbf{b}])$ and $[\mathbf{a}] = \mathcal{C}_{\mathbb{X}} \circ \mathcal{C}_{\overline{\mathbb{X}}}([\mathbf{b}])$
3) The union of contracted boxes (yellow) encloses the part of the boundary $\partial\mathbb{X}$ of $\mathbb{X}$ which is inside $[\mathbf{x}_0]$. Only yellow boxes need to be memorized in the final paving, with the initial green box $[\mathbf{x}_0]$.
4) The contracted boxes form a binary tree. For instance $[\mathbf{a}]$ has two sons: $[\mathbf{d}]$, $[\mathbf{e}]$.
5) If a win-zone is the difference of two boxes. For instance $[\mathbf{x}_0]\backslash[\mathbf{a}]$ is a disconnected win-zone with two colors, and $[\mathbf{b}]\backslash[\mathbf{d}]$ has a single color with an L-shape.
6) In the 2D case, a win-zone may have the shape of an L, a U, or an 0.

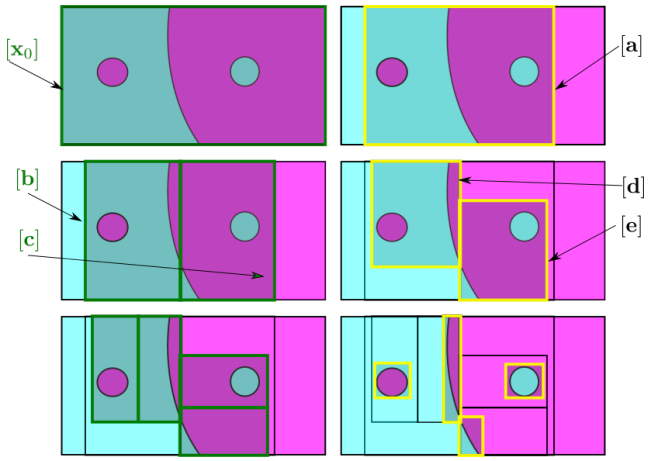

Fig. 1. A contractor-based paver to characterize a set

The main contributions of this paper are the following

1) For existing contractor-based algorithms, the contractors $\mathcal{C}_{\mathbb{X}}$ and $\mathcal{C}_{\overline{\mathbb{X}}}$ are called independently. We will show that they could work in a collaborative manner by considering the constraint associated to the boundary of $\mathbb{X}$. This makes the approach more efficient.
2) A boundary approach allows us to compute the boundary of $\mathbb{X}$ but it remains to know which part of the search space is inside or outside $\mathbb{X}$. We will propose a new method to propagate backward the information corresponding to the color of the win-zone.

Figure 2, illustrates what we want to compute. We first enclose $\partial\mathbb{X}$ inside the yellow boxes. The non-yellow zone (or the win zone) has several connected components. Each of these components is either inside $\mathbb{X}$ or outside. This status corresponds to a binary information: the *color* (magenta: inside or blue: outside). This color will be memorized in the faces of the yellow boxes, as shows on Figure 2, right. If a face has a given color, all other faces associated to the same connected component have the same color. Moreover, if a yellow box has two different colors, we know that it is crosses by the boundary of the solution set. This property is interesting when we deal with problems involving quantifiers. For instance, in Figure 2, we know that $\forall x_2 \in [x_2], \exists x_1 \in [x_1], (x_1, x_2) \in \partial\mathbb{X}$.

The main difficulty remains to get the color of the yellow faces, via a backward propagation. This will be explained in the following section. Note that since it is impossible to go from the inside of a set to the outside without crossing its boundary, we observe in the picture that a magenta zone (corresponding the inside) is always separated from the blue (the outside) by a yellow zone (containing the boundary).

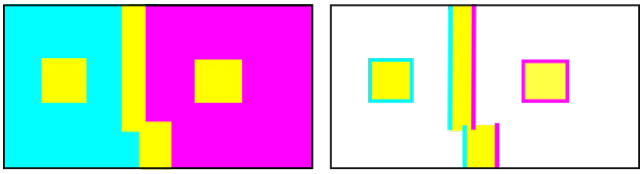

Fig. 2. The yellow boxes and the color faces correspond to what we want to compute

III. Forward-Backward Contraction

A. Principle

We recall here a result given in [18] which can be see as an abstraction of the algorithm presented in [21]. Consider the constraint

$$\mathbf{f}(\mathbf{x}) \in \mathbb{Y},\ \mathbf{x} \in \mathbb{X}(0) \tag{4}$$

where $\mathbf{f}$ is a composition of functions: $\mathbf{f} = \mathbf{f}_n \circ \cdots \circ \mathbf{f}_2 \circ \mathbf{f}_1$. The forward-backward sequence given by Algorithm 1 computes exactly the set $\mathbb{X} = \mathbb{X}(0) \cap \mathbf{f}^{-1}(\mathbb{Y})$.

Algorithm 1 Forward-Backward sequence

Input: $\mathbb{X}(0)$
For $k = 1$ to n
  $\mathbb{X}(k) = \mathbf{f}_k(\mathbb{X}(k-1))$
$\overleftarrow{\mathbb{X}}(n) = \mathbb{Y} \cap \mathbb{X}(n)$
For $k = n$ to 1
  $\overleftarrow{\mathbb{X}}(k-1) = \mathbb{X}(k-1) \cap \mathbf{f}_k^{-1}(\overleftarrow{\mathbb{X}}(k))$
Return $\overleftarrow{\mathbb{X}}(0)$

For each k, we have

$$\begin{array}{rcl} \mathbb{X}(k) & = & \mathbf{f}_{1:k}(\mathbb{X}(0)) \\ \overleftarrow{\mathbb{X}}(k) & = & \mathbf{f}_{1:k}(\mathbb{X}(0)) \cap \mathbf{f}_{k+1:n}^{-1}(\mathbb{Y}) \\ \mathbb{X} & = & \overleftarrow{\mathbb{X}}(0) \end{array} \tag{5}$$

where $\mathbf{f}_{k:\ell} = \mathbf{f}_\ell \circ \cdots \circ \mathbf{f}_k$. The proof of this result is a consequence of the fact that the chain structure of the constraint is also a tree. The forward-backward propagation thus introduces no pessimism.

The computation given by Algorithm 1 cannot be implemented in the computer in the present form. Only a guaranteed outer approximation of this sequence can be implemented in the computer. We need to enclose the sets $\overleftarrow{\mathbb{X}}(k), \mathbb{X}(k)$ in other sets which can be represented and handled numerically [13]. This leads to a branch of computer science named *abstract interpretation* [9].

B. Directed contractor

A *directed contractor* $\mathcal{C}$ for the constraint $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is an operator

$$\mathcal{C} : ([\mathbf{x}], [\mathbf{y}]) \to \left(\overrightarrow{\mathcal{C}}([\mathbf{x}]), \overleftarrow{\mathcal{C}}([\mathbf{x}], [\mathbf{y}]) \right) \tag{6}$$

such that

$$\begin{array}{ccc} \mathbf{f}([\mathbf{x}]) & \subset & \overrightarrow{\mathcal{C}}([\mathbf{x}]) \\ \mathbf{f}^{-1}([\mathbf{y}]) \cap [\mathbf{x}] & \subset & \overleftarrow{\mathcal{C}}([\mathbf{x}], [\mathbf{y}]) \subset [\mathbf{x}] \end{array} \tag{7}$$

Moreover, we should have the monotonicity property:

$$\left\{ \begin{array}{l} [\mathbf{a}] \subset [\mathbf{x}] \\ [\mathbf{b}] \subset [\mathbf{y}] \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \overrightarrow{\mathcal{C}}([\mathbf{a}]) \subset \overrightarrow{\mathcal{C}}([\mathbf{x}]) \\ \overleftarrow{\mathcal{C}}([\mathbf{a}], [\mathbf{b}]) \subset \overleftarrow{\mathcal{C}}([\mathbf{x}], [\mathbf{y}]) \end{array} \right. \tag{8}$$

There exists a directed contractor for $\mathbf{y} = \mathbf{f}(\mathbf{x})$ which is minimal with respect to the inclusion.

This definition is similar to that given in [7] except that here, the definition has been specialized to the specific constraint $\mathbf{y} = \mathbf{f}(\mathbf{x})$ with the input $\mathbf{x}$ and the output $\mathbf{y}$. This will allow us to connect the contractors as a chain (see Figure 3) and to provide the forward-backward sequence.

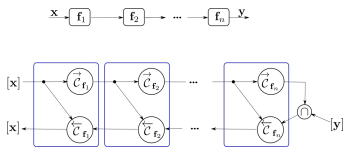

Fig. 3. The directed contractors associated to the constraint $\mathbf{y} = \mathbf{f}_n \circ \cdots \circ \mathbf{f}_2 \circ \mathbf{f}_1(\mathbf{x})$ are connected to form a chain

Example 1. Take the function $f(x_1, x_2) = x_1 + x_2$. The minimal directed contractor for the constraint $y = f(\mathbf{x})$ is given by

$$\begin{array}{ccc} \overrightarrow{\mathcal{C}}([\mathbf{x}]) & = & [x_1] + [x_2] \\ \overleftarrow{\mathcal{C}}([\mathbf{x}], [y]) & = & \begin{pmatrix} [x_1] \cap ([y] - [x_2]) \\ [x_2] \cap ([y] - [x_1]) \end{pmatrix} \end{array} \tag{9}$$

A closed-form minimal contractor is in general not available except for specific functions such as those that are linear or elementary functions (such as $\sin, \cos, \ln, \sqrt{\cdot}$,). A function $\mathbf{f}$ for which such a closed-form expression for the contractor exists is said to be *contractible*.

Example 2. Since a closed-form minimal contractor for the constraint

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x} \tag{10}$$

is

$$\mathcal{C} \begin{pmatrix} [\mathbf{x}] \\ [\mathbf{y}] \end{pmatrix} = \begin{pmatrix} \overrightarrow{\mathcal{C}}([\mathbf{x}]) \\ \overleftarrow{\mathcal{C}}([\mathbf{x}], [\mathbf{y}]) \end{pmatrix} = \begin{pmatrix} \mathbf{A} \cdot [\mathbf{x}] \\ \mathbf{A}^{-1} \cdot [\mathbf{y}] \cap [\mathbf{x}] \end{pmatrix} \tag{11}$$

the function $\mathbf{f}(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x}$ is contractible. The operation given in the expression of $\mathcal{C}$ uses interval arithmetic introduced by Moore [22].

All elementary functions such as $\sin(x), x^2, x_1 + x_2, x_1 \cdot x_2$ are contractible.

C. Contractible decomposition

A *contractible decomposition* of a function $\mathbf{f}$ has the form

$$\mathbf{f} = \mathbf{f}_n \circ \cdots \circ \mathbf{f}_2 \circ \mathbf{f}_1 = \mathbf{f}_{1:n} \tag{12}$$

where each $\mathbf{f}_i$ is contractible.

Note that some functions may not have a contractible decomposition. For instance, the Fresnel integral which arise in the description of near-field Fresnel diffraction phenomena, given by

$$f(x) = \int_0^x \sin \tau^2 \cdot d\tau \tag{13}$$

has no contractible decomposition since it cannot be decomposed as a finite composition of contractible functions.

Example 3. The function

$$f(x_1, x_2) = (x_1 + 2x_2)^2 + (x_1 - x_2)^2 \tag{14}$$

can be decomposed in different ways as illustrated by Figure 4.

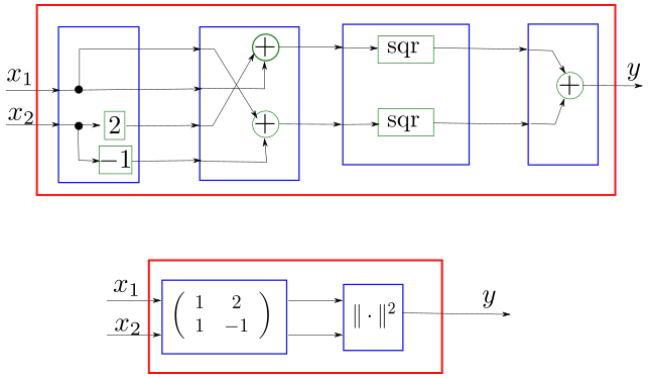

Fig. 4. Scalar contractible decomposition (top); Vector contractible decomposition (bottom)

The first decomposition is the scalar decomposition classically used in the *HC4-revise* contractor [4], [1], [26]. The second decomposition is a vector decomposition, which more efficient. This gain of efficiency is illustrated by Figure 5 where a paver is used to characterize the solution set of the equation $f(x_1, x_2) = 1$. The blue boxes are obtained using the scalar contractible decomposition obtained from the initial box $[-10, 10] \times [-10, 10]$ whereas the red box is obtained directly at the first contraction from the vector contractible decomposition.

D. Forward-backward contractor

Consider the constraint

$$\mathbf{f}(\mathbf{x}) \in \mathbb{Y}, \mathbf{x} \in [\mathbf{x}](0) \tag{15}$$

where $\mathbf{f}$ can be decomposed into a contractible chain: $\mathbf{f} = \mathbf{f}_n \circ \cdots \circ \mathbf{f}_2 \circ \mathbf{f}_1$. Algorithm 2 computes a box enclosure for $\mathbb{X} = [\mathbf{x}](0) \cap \mathbf{f}^{-1}(\mathbb{Y})$. See [4] [26] [1] for more details. It can be seen as an abstract counterpart of the concrete Algorithm 1. This algorithm uses the directed contractor $\left(\overrightarrow{\mathcal{C}}_k, \overleftarrow{\mathcal{C}}_k \right)$ associated to the function $\mathbf{f}_k$.

Fig. 5. Illustration of the gain of efficiency when we use the vector contractible decomposition

Algorithm 2 Forward-backward contractor for $\mathbf{f}(\mathbf{x}) \in \mathbb{Y}$

	Input: $[\mathbf{x}](0)$
1	For $k = 1$ to n
2	$\quad [\mathbf{x}](k) = \overrightarrow{\mathcal{C}}_k([\mathbf{x}](k-1))$
3	$[\mathbf{a}](n) = [\mathbb{Y} \cap [\mathbf{x}](n)]$
4	For $k = n$ to 1
5	$\quad [\mathbf{a}](k-1) = \overleftarrow{\mathcal{C}}_k([\mathbf{x}](k-1), [\mathbf{a}](k))$
	Return $[\mathbf{a}](0)$

The algorithm (which is a *contractor*) is decomposed into two main steps:

- the forward contractions (lines 1,2) where the uncertainty is propagated forward from $\mathbf{x}(0)$ to $\mathbf{x}(n)$;
- the backward contractions (lines 3,4,5) where the uncertainty is propagated backward from $\mathbb{Y}$ to $\mathbf{x}(0)$.

It returns a box $[\mathbf{a}](0)$ which contains the set $\{\mathbf{x} \in [\mathbf{x}](0) \,|\, \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\}$.

At Steps 2 and 5, the contractions are minimal since the functions $\mathbf{f}_k$ are contractible. When we want to find an inner and an outer approximations, we have to run the contractor given by Algorithm 2 twice: once for both $\mathbb{Y}$ and once for its complementary $\overline{\mathbb{Y}}$ [16]. This is clearly not optimal and it is what we want to avoid in this paper.

IV. A boundary approach

A. Principle

The boundary $\partial\mathbb{X}$ of a subset $\mathbb{X}$ of $\mathbb{R}^n$ is the set of points which can be approached both from the inside of $\mathbb{X}$ and from the outside of $\mathbb{X}$. It is also the set of points in the closure of $\mathbb{X}$ not belonging to its interior. We have the following proposition.

Proposition 4. *Consider a continuous function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^p$ *defined everywhere. If* $\mathbb{X} = \mathbf{f}^{-1}(\mathbb{Y})$*, we have*

$$\partial\mathbb{X} \subset \mathbf{f}^{-1}(\partial\mathbb{Y}). \tag{16}$$

Proof: We will show that if $\mathbf{x} \in \partial\mathbb{X}$ then $\mathbf{y} = \mathbf{f}(\mathbf{x}) \in \partial\mathbb{Y}$. Denote by $\mathbb{B}_\eta(\mathbf{x})$ the open ball with radius η and center $\mathbf{x}$. We have

$$\begin{aligned} \mathbf{x} \in \partial\mathbb{X} &\Leftrightarrow \forall\eta > 0, \exists \mathbf{a}, \mathbf{b} \in \mathbb{B}_\eta(\mathbf{x}) | \mathbf{a} \in \mathbb{X}, \mathbf{b} \notin \mathbb{X} \\ &\Leftrightarrow \forall\eta > 0, \exists \mathbf{a}, \mathbf{b} \in \mathbb{B}_\eta(\mathbf{x}) | \mathbf{f}(\mathbf{a}) \in \mathbb{Y}, \mathbf{f}(\mathbf{b}) \notin \mathbb{Y} \\ &\Leftrightarrow \forall\eta > 0, \mathbf{f}(\mathbb{B}_\eta(\mathbf{x})) \cap \partial\mathbb{Y} \neq \emptyset \end{aligned}$$

Now, since $\mathbf{f}$ in continuous and since $\mathbf{y} \in \mathbb{B}_\eta(\mathbf{x})$, for all $\epsilon > 0$, $\exists\eta > 0, \mathbf{f}(\mathbb{B}_\eta(\mathbf{x})) \subset \mathbb{B}_\varepsilon(\mathbf{y})$. Thus

$$\forall\varepsilon > 0, \mathbb{B}_\varepsilon(\mathbf{y}) \cap \partial\mathbb{Y} \neq \emptyset. \tag{17}$$

We conclude that $\mathbf{y} \in \partial\mathbb{Y}$. ■

Remark 5. Proposition 4 requires that f is defined everywhere. Take for instance $f(x) = \sqrt{x^2 - 1}$ and $\mathbb{Y} = [-2, 2]$. We have

$$\begin{aligned} \partial\mathbb{X} &= \partial f^{-1}(\mathbb{Y}) \\ &= \partial([-\sqrt{5}, -1] \cup [1, \sqrt{5}]) \\ &= \{-\sqrt{5}, -1, 1, -\sqrt{5}\} \\ f^{-1}(\partial\mathbb{Y}) &= f^{-1}(\{-2, 2\}) = f^{-1}(\{2\}) \\ &= \{-\sqrt{5}, -\sqrt{5}\} \end{aligned} \tag{18}$$

and the inclusion is not satisfied.

Moreover, Proposition 4 only claims the inclusion $\partial\mathbb{X} \subset \mathbf{f}^{-1}(\partial\mathbb{Y})$ and not the equality. Take for instance $f(x) = x^2$ and $\mathbb{Y} = [0, 4]$, We have

$$\begin{aligned} \partial\mathbb{X} &= \partial f^{-1}(\mathbb{Y}) \\ &= \partial([-2, 2]) \\ &= \{-2, 2\} \\ f^{-1}(\partial\mathbb{Y}) &= f^{-1}(\{0, 4\}) \\ &= \{-2, 0, 2\} \end{aligned} \tag{19}$$

This is consistent with the inclusion, but not with the equality.

Proposition 4 with the hypothesis are illustrated by Figure 6. The set $\mathbb{Y}$ is here a closed disk and $\mathbb{X} = \mathbf{f}^{-1}(\mathbb{Y}) = \{1\} \cup [4, 6] \cup [7, 8]$. The domain of $\mathbf{f}$ is $]-\infty, 2] \cup [3, 6] \cup [7, \infty]$. We have $\mathbf{f}^{-1}(\partial\mathbb{Y}) = \{1, 4, 5, 8\}$ and $\partial\mathbb{X} = \{1, 4, 6, 7, 8\}$. Due to the fact that $\mathbf{f}$ is not defined everywhere some elements of $\partial\mathbb{X}$ are not in $\mathbf{f}^{-1}(\partial\mathbb{Y})$.

Fig. 6. Illustration of Proposition 4

A consequence of Proposition 4 is that, when all hypothesis are fulfilled, we can use a set inversion algorithm to compute an enclosure of the boundary of a set defined as a set inversion problem.

Assume that $\mathbf{f} = \mathbf{f}_n \circ \cdots \circ \mathbf{f}_2 \circ \mathbf{f}_1$ is a contractible decomposition of $\mathbf{f}$. To characterize the set $\mathbb{X} = \mathbb{X}(0) \cap \mathbf{f}^{-1}(\mathbb{Y})$, we

propose to characterize its boundary, as motivated in Section II. In this context, we can apply the boundary-based forward-backward sequence and get Algorithm 3.

Algorithm 3 Boundary forward-backward sequence

Input: $\mathbb{X}(0)$
For $k = 1$ to n
  $\mathbb{X}(k) = \mathbf{f}_k(\mathbb{X}(k-1))$
$\mathbb{A}(n) = \partial\mathbb{Y} \cap \mathbb{X}(n)$
For $k = n$ to 1
  $\mathbb{A}(k-1) = \mathbb{X}(k-1) \cap \mathbf{f}_k^{-1}(\mathbb{A}(k))$
Return $\mathbb{A}(k-1)$

For each k, we have $\mathbb{X}(k) = \mathbf{f}_{0:k}(\mathbb{X}(0))$ and $\mathbb{A}(k) = \mathbf{f}_{0:k}(\mathbb{X}(0)) \cap \mathbf{f}_{k+1:n}^{-1}(\partial\mathbb{Y})$. Therefore $\mathbb{A}(0)$ contains the boundary of $\mathbb{X}(0) \cap \mathbf{f}^{-1}(\mathbb{Y})$. As explained in Subsection III-D, we can approximate the nested pair $(\mathbb{X}(k), \mathbb{A}(k))$ by two boxes $[\mathbf{x}](k), [\mathbf{a}](k)$ such that

$$\begin{array}{ccc} [\mathbf{x}](k) & \supset & \mathbb{X}(k) \\ [\mathbf{a}](k) & \supset & [\mathbf{x}](k) \cap \mathbf{f}_{k+1:n}^{-1}(\partial\mathbb{Y}) \end{array} \tag{20}$$

As illustrated by Figure 7, we see that $[\mathbf{a}](k)$ contains more than $\mathbb{A}(k)$. Indeed, it is important that the set $[\mathbf{x}](k) \backslash [\mathbf{a}](k) = \{\mathbf{x} \in [\mathbf{x}](k) | \mathbf{x} \notin [\mathbf{a}](k)\}$ does not contain a single point of the boundary $\mathbf{f}_{k+1:n}^{-1}(\partial\mathbb{Y})$.

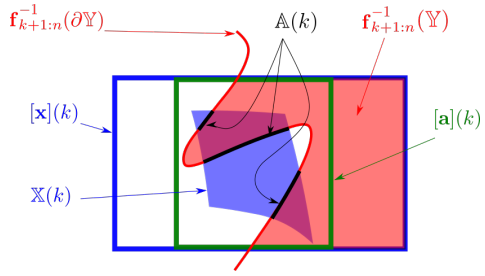

Fig. 7. Approximation by boxes of sets $\mathbb{A}(k)$ and $\mathbb{X}(k)$

An illustration of the boundary-based forward-backward propagation is given by Figure 8.

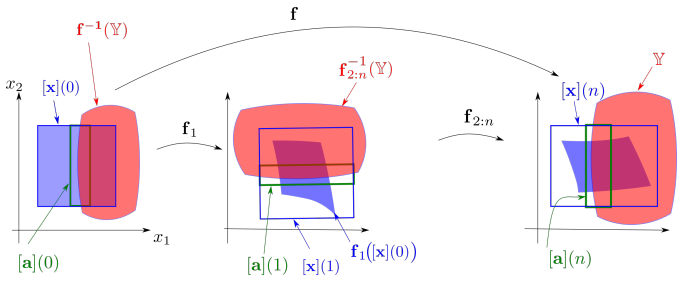

Fig. 8. Forward-backward boundary propagation

Characterizing the boundary is not sufficient to solve the set inversion problem, we also need to determine the colors of the faces of the boxes $[\mathbf{a}](k)$ as motivated by Section II.

B. Cardinal directions

The set $\mathbb{R}^n$ has $2n$ cardinal directions which can be represented by the set

$$\mathcal{D} = \{\lambda = (i, \sigma) | i \in \{1, n\}, \sigma \in \{-, +\}\} \tag{21}$$

The box

$$[\mathbf{x}] = [x_1^-, x_1^+] \times \cdots \times [x_n^-, x_n^+]. \tag{22}$$

has n interval components, each of them defined by two bounds. We thus have $2n$ bounds to define $[\mathbf{x}]$. Each of these bounds corresponds to one cardinal direction. If λ is a cardinal direction, we denote by x^λ the corresponding bound of $[\mathbf{x}]$. Each face of $[\mathbf{x}]$, denoted by $[\mathbf{x}]^\lambda$, is also associated to one cardinal direction. Define by $\mathcal{H}_\lambda([\mathbf{x}])$ the half space made with all points that can see the λ^{th} face of $[\mathbf{x}]$ from the outside.

Example 6. Consider the cardinal direction the $\lambda = (2, -)$ and the box

$$[\mathbf{x}] = [1, 2] \times [3, 4] \times [5, 6]. \tag{23}$$

We have $x^\lambda = 3$, the associated face is $[1, 2] \times [3, 3] \times [5, 6]$ and $\mathcal{H}_\lambda([\mathbf{x}]) = \{\mathbf{x} | \, x_2 < 3\}$.

C. Win boxes

Given two boxes $[\mathbf{a}]$ and $[\mathbf{x}]$, with $[\mathbf{a}] \subset [\mathbf{x}]$. For a given $\lambda \in \mathcal{D}$, we define the λth win box, as

$$[\mathbf{x}] \backslash [\mathbf{a}]|_\lambda = [\mathbf{x}] \cap \mathcal{H}_\lambda([\mathbf{a}]) \tag{24}$$

This is illustrated by Figure 9. In Subfigure (1), we have 4 overlapping win boxes, all in the same connected component. In (b), we have two win boxes and two connected components. In (3), we have one win box and in (4), two win boxes.

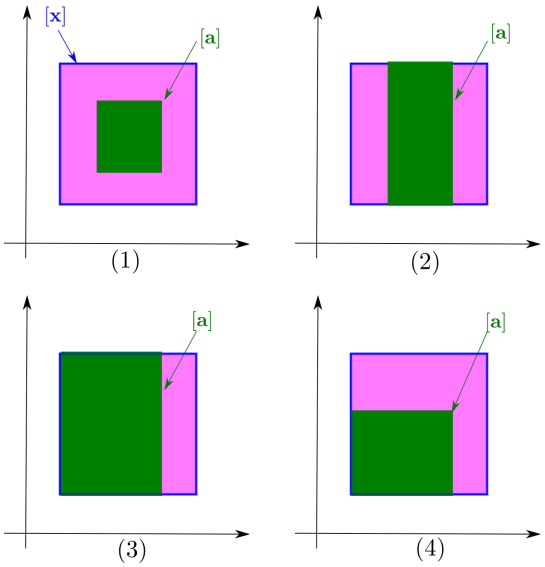

Fig. 9. Illustration of the win boxes (magenta) associated with the difference between $[\mathbf{x}]$ and $[\mathbf{a}]$

D. Color

Given the pair $[\mathbf{x}](k), [\mathbf{a}](k)$ in the algorithm of subsection II. We associate to each bound $a^\lambda(k)$ of $[\mathbf{a}](k)$, the quantity $c(a^\lambda(k)) \in \{0, 1, ?\}$ such that

$$\begin{array}{rcl} c(a^\lambda(k)) = 1 & \Rightarrow & \mathbf{f}_{k+1:n}([\mathbf{x}]^\lambda) \subset \mathbb{Y} \\ c(a^\lambda(k)) = 0 & \Rightarrow & \mathbf{f}_{k+1:n}([\mathbf{x}]^\lambda) \cap \mathbb{Y} = \emptyset \end{array} \tag{25}$$

The quantity $c(a^\lambda(k))$ corresponds to the face color introduced in Section II. If we take the situation of Figure 7, we have $c(a^{(1,-)}(k)) = 0$, $c(a^{(1,+)}(k)) = 1$, $c(a^{(2,-)}(k)) =?$, $c(a^{(2,+)}(k)) =?$.

E. Algorithm

Algorithm 4 is an abstraction of Algorithm 3, with a backward propagation of the color. It allows us to know which part of the search space will be mapped inside $\mathbb{Y}$ and which part will be mapped outside.

Algorithm 4 Forward-backward algorithm with colors

Input: $[\mathbf{x}](0)$
For $k = 1$ to n
  $[\mathbf{x}](k) = \overrightarrow{\mathcal{C}}_k([\mathbf{x}](k-1))$
$[\mathbf{a}](n) = [\partial\mathbb{Y} \cap [\mathbf{x}](n)]$
For all $\lambda \in \mathcal{D}$, $c(a^\lambda(n)) = \begin{cases} 1 & \text{if } [\mathbf{x}]^\lambda(n) \subset \mathbb{Y} \\ 0 & \text{otherwise} \end{cases}$
For $k = n$ to 1
  $[\mathbf{a}](k-1) = \overleftarrow{\mathcal{C}}_k([\mathbf{x}](k-1), [\mathbf{a}](k))$
Return $[\mathbf{a}](0)$

At Step 5, we use the backward contractor $\overleftarrow{\mathcal{C}}_k([\mathbf{x}](k-1), [\mathbf{a}](k))$ given by Algorithm 5.

Algorithm 5 Backward contractor $\overleftarrow{\mathcal{C}}_k([\mathbf{x}], [\mathbf{y}])$

Input: $\mathbf{f}_k, [\mathbf{x}], [\mathbf{y}]$
$[\mathbf{a}] = \left[[\mathbf{x}] \cap \mathbf{f}_k^{-1}([\mathbf{y}])\right]$
For all λ such that $x^\lambda \neq a^\lambda$
  Take λ_1 such that $\mathbf{f}_k([\mathbf{x}]^\lambda) \subset \mathcal{H}_{\lambda_1}([\mathbf{y}])$
  $c(a^\lambda) = c(y^{\lambda_1})$
Return $[\mathbf{a}]$

The principle of the backward operator $\overleftarrow{\mathcal{C}}(\mathbf{f}, [\mathbf{x}], [\mathbf{y}])$ is illustrated by Figure 10.

Step 2. For the cardinal direction $(1,+)$, we have $a_1^+(k-1) < x_1^+(k-1)$. This means that we had a backward contraction with respect to the direction λ and the part that has been contracted is either inside $\mathbf{f}_{k:n}^{-1}(\mathbb{Y})$ or in its complementary.

Step 3. We take the face $[\mathbf{x}]^{(1,+)}(k-1)$ in black, we apply $\mathbf{f}_k$ check on which win box which fall in. Here, it is the win box corresponding to $\lambda_1 = (2,+)$. Note that testing one point is sufficient (see the green point), since all points $[\mathbf{x}]^{(1,+)}(k-1)$ will fall inside the same win box.

Step 4. Since $c(a^{\lambda_1}(k)) = 1$, we get $c(a_1^+(k)) = 1$, which means that the face $[\mathbf{x}]^{(1,+)}(k-1)$ us fully inside $\mathbf{f}_{k:n}^{-1}(\mathbb{Y})$.

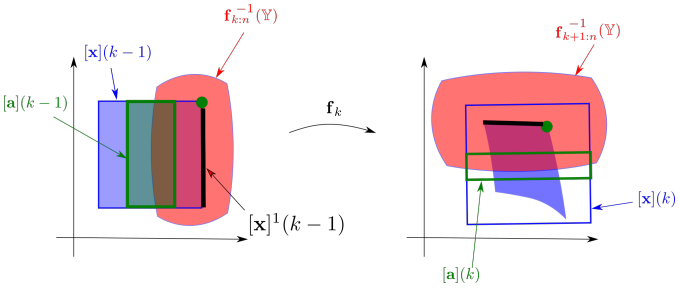

Fig. 10. Backward propagation of the bound colors

F. Example 1: the sum

Consider the function $f(\mathbf{x}) = x_1 + x_2$. The forward step needed at Step 2 of the Algorithm 4 is given by

$$[y] = [x_1] + [x_2]. \tag{26}$$

For the backward step, $\overleftarrow{\mathcal{C}}([\mathbf{x}], [y])$ needed at Step 5 of the Algorithm 4, we have to instantiate Algorithm 5 to our specific case.

Proposition 7. *The backward step (see Algorithm 5) for the constraint $f(\mathbf{x}) = x_1 + x_2$ is:*

$$[\mathbf{a}] = [\mathbf{x}] \cap \begin{pmatrix} [y] - [x_2] \\ [y] - [x_1] \end{pmatrix} \tag{27}$$

and

$$\begin{array}{l} \text{if } x_1^- < a_1^-, \text{ then } c(a_1^-) = c(y^-) \\ \text{if } a_1^+ < x_1^+, \text{ then } c(a_1^+) = c(y^+) \\ \text{if } x_2^- < a_2^-, \text{ then } c(a_2^-) = c(y^-) \\ \text{if } a_2^+ < x_2^+, \text{ then } c(a_2^+) = c(y^+) \end{array}$$

Proof: For $f(\mathbf{x}) = x_1 + x_2$, Algorithm 5 we get

$[a_1] = [x_1] \cap [y] - [x_2]$
$[a_2] = [x_2] \cap [y] - [x_1]$
For all $\lambda \in \{(1,-),(1,+),(2,-),(2,+)\}$ such that $x^\lambda \neq a^\lambda$
  Take λ_1 such that $f([\mathbf{x}]^\lambda) \subset \mathcal{H}_{\lambda_1}([y])$
  $c(a^\lambda) = c(y^{\lambda_1})$

We limit our reasoning to $\lambda = (1,-)$, but for a complete proof, the same should be done for other values of λ. We get at Step 2 that if $x_1^- < a_1^-$ we have to choose $\lambda_1 \in \{-,+\}$. Take $\lambda_1 = -$, we get

$$\begin{array}{rcl} & & f([\mathbf{x}]^\lambda) \subset \mathcal{H}_{\lambda_1}([y]) \\ \Leftrightarrow & & x_1^- + [x_2] \subset [-\infty, y^-] \\ \Leftrightarrow & & x_1^- + x_2^+ \leq y^- \end{array} \tag{28}$$

Now, since from Step 1 we have $a_1^- = \max(x_1^-, y^- - x_2^+)$ and since $x_1^- < a_1^-$, we get $a_1^- = y^- - x_2^+$. The condition (28) becomes $x_1^- + x_2^+ \leq a_1^- + x_2^+$ which is always true. The condition of Step 3 is thus always true as soon as we take $\lambda_1 = -$ if $\lambda \in \{(1,-),(2,-)\}$and $\lambda_1 = +$ if $\lambda \in \{(1,+),(2,+)\}$. The instantiation for $c(a^\lambda)$ at Step 4 is straightforward. ■

G. Example 2. The square function

Consider the function $f(x) = x^2$. The forward step needed at Step 2 of the Algorithm 4 is given by

$$[y] = [x]^2$$

For the backward step, we have the following proposition.

Proposition 8. *The backward step (see Algorithm 5) for $f(x) = x^2$, translates into*

$$\begin{array}{l}[a] = \left[\{x \in [x],\, x^2 \in [y]\}\right] \\ \text{if } x^- < a^-, \\ \quad \text{if } x^{-2} < y^-, \text{ then } c(a^-) = c(y^-) \\ \quad\quad \text{else } c(a^-) = c(y^+) \\ \text{if } a^+ < x^+, \\ \quad \text{if } x^{+2} < y^-, \text{ then } c(a^+) = c(y^-) \\ \quad\quad \text{else } c(a^+) = c(y^+)\end{array}$$

Proof: For $f(x) = x^2$, Algorithm 5 becomes

1	$[a] = \left[\{x \in [x],\, x^2 \in [y]\}\right]$
2	For all $\lambda \in \{-,+\}$ such that $x^\lambda \neq a^\lambda$
3	Take λ_1 such that $(x^\lambda)^2 \subset \mathcal{H}_{\lambda_1}([y])$
4	$c(a^\lambda) = c(y^{\lambda_1})$

Take $\lambda = -$. We get at Step 2 that if $x^- < a^-$ we have to choose $\lambda_1 \in \{-,+\}$ such that $x^{-2} \subset \mathcal{H}_{\lambda_1}([y])$. For $\lambda_1 = -$, the condition at Step 4 translates into $x^{-2} \leq y^-$ and in this case, we get at Step 4, $c(a^-) = c(y^-)$. It the condition $x^{-2} \leq y^-$ is false, it means that $\lambda_1 = +$ satisfies the condition and thus $c(a^-) = c(y^+)$.

Take $\lambda = +$. We get at Step 2 that if $x^+ > a^+$ we have to choose $\lambda_1 \in \{-,+\}$ such that $x^{+2} \subset \mathcal{H}_{\lambda_1}([y])$. For $\lambda_1 = -$, the condition at Step 4 is $x^{+2} \leq y^-$ and if satisfies, we get at Step 4, $c(a^+) = c(y^-)$. If the condition is false, it means that $\lambda_1 = +$ satisfies the condition and thus $c(a^+) = c(y^+)$. ■

V. Test-cases

In this Section, we give two examples. The first example illustrates an academic set inversion problem involving a nonlinear function. The Python code is also given. The second example is a localization problem using a TDOA technique (Time Difference Of Arrival). The problem is known to be ill-conditioned and highly nonlinear [20].

A. A simple illustrative example

Consider the set inversion problem

$$\mathbb{X} = \begin{pmatrix} [-3,3] \\ [-3,3] \end{pmatrix} \cap \mathbf{f}^{-1} \begin{pmatrix} [-1,2] \\ [1,4] \end{pmatrix} \tag{29}$$

where

$$\mathbf{f}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ (x_1+x_2)^2 \end{pmatrix}. \tag{30}$$

The function $\mathbf{f}$ has the following contractible decomposition:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} s = x_1 + x_2 \end{pmatrix} \rightarrow \begin{pmatrix} s \\ s^2 \end{pmatrix}. \tag{31}$$

This decomposition leads us to a forward-backward boundary contractor. Combined with a paver, we get the paving of Figure 11. For this example, we have the inner and outer approximation at the same price as if we characterize the boundary only. The Python code associated to this example is available at https://www.ensta-bretagne.fr/jaulin/sepbox.html.

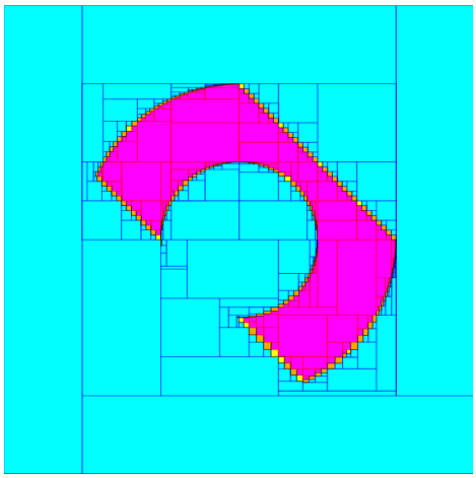

Fig. 11. Set inversion problem solved using a boundary approach

B. Test case 2. TDOA

Pseudorange multilateration (also termed *hyperbolic positioning*) is a technique for determining a robot position based on measurement of the *times of arrival* of signals (radio or acoustic) having a known speed when propagating from the stations to the robot. These stations are at known locations and have synchronized clocks, but these clocks are not synchronized with the clock of the robot. Consequently, the robot do not measure the distance to the stations. Instead, it measures the difference of distances between its position and that of the stations. The problem is known as TDOA (*time difference of arrivals*) and is known to be very ill-conditioned [20].

As an example, consider one robot in the plane where three stations at known positions $(a_i, b_i), i \in \{1,2,3\}$ emit at the same unknown time a sound, as illustrated by Figure 12. The three corresponding sounds are received at times t_i, as given by the following table.

i	1	2	3
a_i	4	13	16
b_i	6	7	10
t_i	15	23	27

We assume for simplicity that speed of the sound is $1m.s^{-1}$. Since the emission time is not known by the robot, we cannot translate these times into distances. However, the difference of times is directly related to difference of distances (called the *pseudo distances*).

The measured pseudo-distance are thus

$$\begin{array}{rcl} y_1 & = & 23 - 15 = 8 \\ y_2 & = & 27 - 23 = 4 \end{array} \tag{32}$$

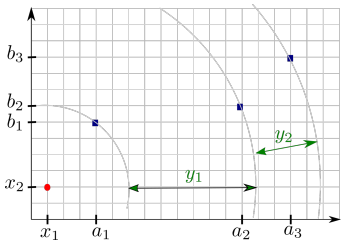

Fig. 12. The robot (red) measures the pseudo distances y_1, y_2 to the stations (blue)

We assume that the accuracy of the pseudo distance measurements is $\varepsilon = 0.001$. The set $\mathbb{X}$ of all feasible location vectors is defined by

$$\mathbf{f}(\mathbf{x}) \in [8-\varepsilon, 8+\varepsilon] \times [4-\varepsilon, 4+\varepsilon] \tag{33}$$

where

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \sqrt{(13-x_1)^2+(7-x_2)^2} - \sqrt{(4-x_1)^2+(6-x_2)^2} \\ \sqrt{(16-x_1)^2+(10-x_2)^2} - \sqrt{(13-x_1)^2+(7-x_2)^2} \end{pmatrix}$$

A decomposition in contractible function of $\mathbf{f}(\mathbf{x})$ is given by Figure 13.

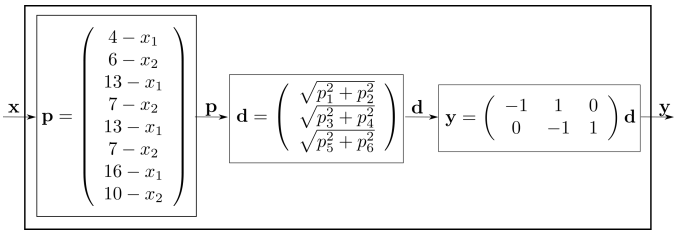

Fig. 13. Decomposition into contractible functions

Figure 14 presents the results obtained using a paver with two different contractors, for the same accuracy. The left figure is obtained using a classical forward backward contractor which generated 90841 boxes. Our boundary based contractor with the contractible decomposition generated only 35586 boxes.

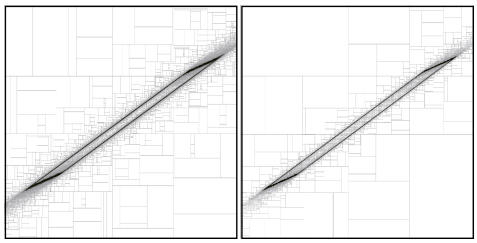

Fig. 14. Left: characterization of the solution set $\mathbb{X}$ using a classical forward-backward contractor;
Right: the same using a boundary approach
For both, the frame box is $[0.94, 1.06] \times [1.94, 2.06]$

VI. Conclusion

In this paper, we have proposed a new forward-backward contraction procedure for set inversion. The principle is to inverse the boundary of the set $\mathbb{Y}$ to be inverted and to retro-propagate a color on the interval bounds in order to decide if the win boxes are inside or outside the solution set.

This approach has several interesting features:

1) Contrarily to existing contractor-based set inversion algorithm, we do not have to retro-propagate twice (once for inside $\mathbb{Y}$, once for outside), but only once for the boundary.
2) The cost of retro-propagating the color is almost negligible since it requires only comparisons between floating point numbers.
3) When a box has two faces of two different colors, it means that it is crossed by the boundary of the solution set. This information which can be useful, for projection problems [15] and quantified problems [12], was not provided by existing approaches.
4) The procedure can easily be implemented in an HC4-revised procedure where the function to be inverted is represented by a DAG (directed Acyclic Graphs). There exists some efficient contractors built for equations that can be used for inverting a boundary of $\mathbb{Y}$. This is the case of the Newton contractor [22], the monotonicity based contractor [2], the centered-form based contractors, etc. We still need to adapt our procedure to retro-propagate the colors at a minimal cost, as we have done for contractors for chains of contractible functions.

References

[1] I. Araya, B. Neveu, and G. Trombettoni. Exploiting Common Subexpressions in Numerical CSPs. In *Proc. CP, Constraint Programming*, pages 342–357, LNCS 5202, 2008.

[2] I. Araya, G. Trombettoni, and B. Neveu. Exploiting monotonicity in interval constraint propagation. In *AAAI*, pages 9–14, 2010.

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

[4] F. Benhamou, F. Goualard, L. Granvilliers, and J-F. Puget. Revising Hull and Box Consistency. In *ICLP*, pages 230–244, 1999.

[5] F. Benhamou, F. Goualard, E. Languenou, and M. Christie. An algorithm to compute inner approximations of relations for interval constraints. In *Conference on Perspectives of Systems Informatics (PSI'99)*, 1999.

[6] M. Cébério and L. Granvilliers. Solving nonlinear systems by constraint inversion and interval arithmetic. In *Artificial Intelligence and Symbolic Computation*, volume 1930, pages 127–141, LNCS 5202, 2001.

[7] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.

[8] E. Colle and Galerne. Mobile robot localization by multi-angulation using set inversion. *Robotics and Autonomous Systems*, 61(1):39–48, 2013.

[9] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by

construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977.

[10] D. Daney, N. Andreff, G. Chabert, and Y. Papegay. Interval Method for Calibration of Parallel Robots : Vision-based Experiments. *Mechanism and Machine Theory, Elsevier*, 41:926–944, 2006.

[11] V. Drevelle and P. Bonnifait. Localization confidence domains via set inversion on short-term trajectory. *IEEE Transactions on Robotics*, 2013.

[12] A. Goldsztejn and L. Jaulin. Inner and outer approximations of existentially quantified equality constraints. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming, (CP 2006)*, Nantes (France), 2006.

[13] E. Goubault and S. Putot. Static analysis of numerical algorithms. In *In Proceedings of SAS 06, LNCS 4134*, pages 18–34. Springer-Verlag, 2006.

[14] E. R. Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker, New York, NY, 1992.

[15] M. Hladík and S. Ratschan. Efficient Solution of a Class of Quantified Constraints with Quantifier Prefix Exists-Forall. *Mathematics in Computer Science*, 8(3-4):329–340, July 2014.

[16] L. Jaulin and B. Desrochers. Introduction to the algebra of separators with application to path planning. *Engineering Applications of Artificial Intelligence*, 33:141–147, 2014.

[17] L. Jaulin, J. L. Godet, E. Walter, A. Elliasmine, and Y. Leduff. Light scattering data analysis via set inversion. *Journal of Physics A: Mathematical and General*, pages 7733–7738, 1997.

[18] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed nonlinear estimation using constraint propagation on sets. *International Journal of Control*, 74(18):1772–1782, 2001.

[19] V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. Computational complexity and feasibility of data processing and interval computations. *Reliable Computing*, 4(4):405–409, 1997.

[20] H. B. Lee. Accuracy limitations of hyperbolic multilateration systems. *IEEE Transactions on Aerospace and Electronic Systems*, AES-11(1):16–29, 1975.

[21] U. Montanari and F. Rossi. Constraint relaxation may be perfect. *Artificial Intelligence*, 48(2):143–170, 1991.

[22] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA, 1979.

[23] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, 2004.

[24] S. Rohou, L. Jaulin, L. Mihaylova, F. Le Bars, and S. Veres. *Reliable robot localization*. ISTE Group, 2019.

[25] M. van Emden. Algorithmic power from declarative use of redundant constraints. *Constraints*, 4(4):363–381, 1999.

[26] P. van Hentenryck, Y. Deville, and L. Michel. *Numerica: A Modeling Language for Global Optimization*. MIT Press, Boston, MA, 1997.

[27] P. Herrero Vinas, M. A. Sainz, J. Vehi, and L. Jaulin. Quantified set inversion algorithm with applications to control. *Reliable computing*, 11(5):369–382, 2006.