UV 5.4

Stage assistant ingénieur

Motion planning using interval analysis

Benoît DESROCHERS

Supervisor :
Luc Jaulin

March 12, 2014

Abstract: Interval analysis is a powerful tool which could deal with complex problems especially when there are non linear or there have no solution. However, motion planning using intervals remains a subject that has been little studied to date, and tools like *contractor* has never been used for this purpose. This study aim to develop tools and algorithms, which could deal with path panning issues, made as simple as possible, in order to simplify the usage of interval in this field of research. Throw an example of application : the wire loop game, we illustrated the possibility of doing path planning with interval analysis in an efficient way. We also introduced the separator object and the one associated with the constraint: " a point is inside a polygon". Next we were able to show explicitly a feasible path throw the configuration space. The main advantage of interval methods seems to be their capacity to represent the $\mathbb{C}$-space as tree or graph composed of continuous boxes, without loosing any solution, and on which we could apply very efficient graph algorithms. These methods are guaranteed and are able to prove that no path exist. To our knowledge, it is a key point which could distinguish interval methods from the others However they need to be compare with other available ones.

Contents

Introduction

Interval analysis is a powerful tool which could deal with complex problems especially when there are non linear or there have no solution. It is used in a wide range of applications such as state estimation, [13] [5] [10] [1] [11] or calibration [4] [18]. However, motion planning using intervals remains a subject that has been little studied to date, and tools like *contractor* has never been used for this purpose. This study aim to develop tools and algorithms, which could deal with path-planing issues, made as simple as possible, in order to simplify the usage of interval in this field of research.

After generalities about path planning, this document deals with an example of application: wire loop game. The theory needed such as separator and geometrical constraints are introduced in section 2 as well as results. Finally the last section introduces path finding algorithm are described with their corresponding results.

1 Generalities about Path-Planning

1.1 $\mathbb{C}$-space

The terms *motion planning* and *trajectory planning* are often used to describe algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move. Let us consider a system (Σ) and let $\mathbb{C}$-space denote the space of all feasible states for our system. This will be referred to the *configuration space* (C-space), based on Lagrangian mechanics and the seminal work of Lozano-Pérez [14, 16], who extensively utilized this notion in the context of planning. The motion planning literature was further organized around this concept by Latombe's book [12]. Once the configuration space is clearly understood, many motion-planning problems that appear different in terms of geometry and kinematics can be solved by the same planning algorithms. This level of abstraction is therefore very important. An example of such objects are industrial robots which are kinematic chains in which adjacent links are connected by n prismatic or rotary joints, each with one degree of freedom. The position and orientation of each link of the industrial robot can be characterized by n real numbers, which are the coordinates of a single n-dimensional point in $\mathbb{C}$-space (see [15], for more information). Let's take $x_i \in \mathbb{C}$-space as an initial state and $x_g \in \mathbb{C}$-space as a final state, a basic motion planning problem is to produce a continuous motion that connects x_i to x_g while avoiding collision with known obstacles.

Accordingly, determining the $\mathbb{C}$-space or more precisely its subset, $\mathbb{C}_{free}$, of feasible configurations that avoids collision with obstacles is a key feature of the path planning issues.

1.2 Main Motion Planning Algorithms.

1.2.1 Generalities

Low-dimensional problems can be solved with grid-based algorithms that overlay a grid on top of configuration space, or geometric algorithms that compute the shape and connectivity of $\mathbb{C}_{free}$. These methods assume that each configuration is identified with a grid point. At each grid point, the robot is allowed to move to adjacent grid points as long as the line between them is completely contained within $\mathbb{C}_{free}$ (this is tested with collision detection). This discretizes the set of actions, and search algorithms (like A^*) are used to find a path from the start to the goal. However, the number of points on the grid grows exponentially with the dimension of the configuration space, which makes them inappropriate for high-dimensional problems. Nevertheless exact motion planning for high-dimensional systems under complex constraints is computationally intractable. Another way is Potential-field algorithms [9] which are efficient, but fall prey to local minimal (an exception is the harmonic potential fields [3]). Sampling-based algorithms avoid the problem of local minimal, and solve many problems quite quickly by taking a countable number of samples in the C-space. The sampling process is often based on random algorithms like *rapidly exploring random tree* (RRT) [17] or *probabilistic roadmap methods* (PRMs) [8]. The probability of failure decreases to zero as more time is spent on finding the solution. However if no solution exist, these algorithms could run forever. Sampling-based algorithms are currently considered state-of-the-art for motion planning in high-dimensional spaces, and have been applied to problems which have dozens or even hundreds of dimensions (robotic manipulators, biological molecules, animated digital characters, and legged robots).

1.2.2 Interval-Based Search

These approaches are similar to grid-based search approaches except that they generate a paving covering entirely the configuration space instead of a grid. The $\mathbb{C}_{free}$ is decomposed into sub-paving $\mathbb{C}^+$, $\mathbb{C}^-$ such as:

$$\mathbb{C}^- \subset \mathbb{C}_{free} \subset \mathbb{C}^+. \tag{1}$$

Interval analysis could thus be used when $\mathbb{C}_{free}$ cannot be described by linear inequalities in order to have a guaranteed enclosure. The robot is thus allowed to move freely in $\mathbb{C}^-$, and cannot go outside $\mathbb{C}^+$. To both sub-pavings, a neighbor graph is build and graph algorithms such as *Dijkstra* or A^* could be used. When a path is feasible in $\mathbb{C}^-$, it is also feasible in $\mathbb{C}_{free}$. When no path exists in $\mathbb{C}^+$ from one initial configuration to the goal, we have the guarantee that no feasible path exists in $\mathbb{C}_{free}$. However, as for the grid-based approach, the actual interval approach is inappropriate for high-dimensional problems, due to the fact that the number of boxes generated grows exponentially with respect to the dimension of configuration space. Nevertheless, this algorithm is only based on bisection and inclusion test, which are known to be very inefficient when the dimension of

the problem increases, and could be improved by using *Contractor* and a clever bisection's process. The algorithm called *Cameleon* introduced in [6] avoids bisecting boxes which belong to the configuration space and not influence the final result. This method was the base of the current work.

2 Configuration Space for the Wire Loop Game

The problem to be considered is a 2D version of the wire loop game. This game involves a metal loop on a handle and a length of curved wire (see figure 1). The player holds the loop in one hand and attempts to guide it along the curved wire without touching the loop to the wire. In our 2D version of this game the player is an articulated robot with two rotary joints and the loop is a segment. The purpose of this project is to find a safe path which allows the loop to make a full rotation around the wire.

2.1 Modeling

The curved wire is composed of union of oriented segments which form a polygon P. Its border defines two sets : $\mathbb{Y}$ which corresponds of all points inside P (gray in the figure 1) and $\overline{\mathbb{Y}}$ its complementary set. The length of the first and second arms are 4 and 2 respectively. The length of the loop is 1. The feasible configuration space $\mathbb{C}_{free}$ is the set of parameters such as the point $\mathbf{a}$ is inside P and $\mathbf{b}$ is outside P defined by :

$$\mathbb{C}_{free} = \{\mathbf{x} \in [-\pi, \pi]^2,\ f_1(\mathbf{x}) \in \mathbb{Y} \text{ and } f_2(\mathbf{x}) \notin \mathbb{Y}\}$$

where f_1 (resp. f_2) is the non linear function which transform the point $\mathcal{O}(0,0)^T$ into the $\mathbf{a}$ (resp. $\mathbf{b}$). In our case f_l is :

$$f_l(\mathbf{x}) = 4 \begin{pmatrix} \cos(x_1) \\ \sin(x_2) \end{pmatrix} + l \begin{pmatrix} \cos(x_1 + x_2) \\ \sin(x_1 + x_2) \end{pmatrix}$$

with $l \in \{1, 2\}$ and $\mathbf{x} = (x_1, x_2)^T$.

Finally, we obtain a set, defined with constraints, which corresponds the following set inversion :

$$\mathbb{C}_{free} = f_1^{-1}(\mathbb{Y}) \cap f_2^{-1}\left(\overline{\mathbb{Y}}\right)$$

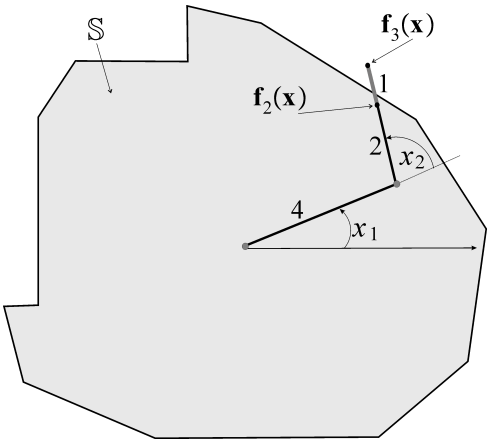


Figure 1: Two-dimensional wire loop game. In a feasible configuration, the gray segment is not allowed to touch the wire. Is it possible to perform to complete circular path

2.2 Separator

Introduction

Set inversion problems aim to bracket a set $\mathbb{X}$, defined by constraints, between two sub-pavings (i.e. union of non overlapping boxes) $\mathbb{X}^-$ and $\mathbb{X}^+$ such that:

$$\mathbb{X}^- \subset \mathbb{X} \subset \mathbb{X}^+$$

We have also $\mathbb{X}^+ = \mathbb{X}^- \cup \Delta\mathbb{X}$ where $\Delta\mathbb{X}$ is the border of the set. The width of this border depends on the accuracy of the estimation. Generally, the set $\mathbb{X}$ is a combination of union, intersection and/or composition of elementary constraints such as $f_i(x) < 0$. For each elementary constraint a contractor $\mathcal{C}_{f_i}$ is associated and with the contractor's arithmetic (see [2]) the global contractor $\mathcal{C}_f$ is easy to build. This contractor approximates the outer sub-paving $\mathbb{X}^+$. If the inner approximation ($\mathbb{X}^-$) is needed, the *De Morgan* rules can be used to express the complementary $\overline{\mathbb{X}}$ set of $\mathbb{X}$ and build the associated contractor $\overline{\mathcal{C}_f}$.

However, apply the *De Morgan* rules for a complex combination of constraints are not easy and need to be done by hand. This is one of the main reasons why *Separators*, a new tool introduced in [7], was developed. It allows us to deal properly and efficiently with complex combinations of inner and outer contractors.

The second reason is that sometimes, equations of the inner and outer part are not available or associated contractors aren't minimal. However, we could have the equation of the border $\delta\mathbb{X}$ represented by a constraint such as $f_b(x) = 0$. In this case, information about which removed parts are inside or outside $\mathbb{X}$ set are lost. *Separator* , which allows us to handle at the same time the inner and outer contractors, provides a new way for building minimal contractors.

After a short overview of Contractors, the *Separator* object will be defined.

2.2.1 Contractor

A contractor is an operator from $\mathbb{IR}^n \longrightarrow \mathbb{IR}^m$ associated to a set $\mathbb{X}$ such as:

$$\begin{array}{ll} \forall [\mathbf{x}] \in \mathbb{IR}^n \ , \ \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}] & (\textit{contractance}) \\ \forall [\mathbf{x}] \in \mathbb{IR}^n \ , \ \mathcal{C}([\mathbf{x}]) \cap \mathbb{X} = [\mathbf{x}] \cap \mathbb{X} & (\textit{completeness}) \end{array}$$

A contractor $\mathcal{C}$ could have the following properties:

$$\begin{array}{rl} \mathcal{C} \text{ is monotonic if} & \forall [\mathbf{x}],\ [\mathbf{y}] \in \mathbb{IR}^n, [\mathbf{x}] \subset [\mathbf{y}] \rightarrow \mathcal{C}([\mathbf{x}]) \subset \mathcal{C}([\mathbf{y}]) \\ \mathcal{C} \text{ is idempotent if} & \forall [\mathbf{x}] \in \mathbb{IR}^n,\ \mathcal{C}(\mathcal{C}([\mathbf{x}])) = \mathcal{C}([\mathbf{x}]) \\ \mathcal{C} \text{ is minimal if} & \forall [\mathbf{x}] \in \mathbb{IR}^n,\ \mathcal{C}([\mathbf{x}]) \cap \mathbb{X} = [\mathbf{x}] \cap \mathbb{X} \end{array}$$

The first property is essential for the set inversion algorithm because there are required for the convergence. Hopefully, there are easy to get. On the contrary, the last property is desirable to increase the efficiency and the algorithm but is much more difficult to obtain. A set $\mathcal{S}$ is consistent with the contractor $\mathcal{C}$ (we will write $\mathcal{S} \sim \mathcal{C}$) if for all $[\mathbf{x}]$, we have :

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

We define the negation $\neg\mathcal{C}$ of a contractor $\mathcal{C}$ as follows :

$$\neg\mathcal{C}([\mathbf{x}]) = \{x \in [\mathbf{x}] | x \notin \mathcal{C}([\mathbf{x}])\}$$

Note that $\neg\mathcal{C}([\mathbf{x}])$ is not a box in general, but a union of boxes.

2.2.2 Definition and Algebra

Implicit Definition

A Separator $\mathcal{S}$ associated with the set $\mathbb{X}$ is an application such as:

$$\begin{array}{rccc} \mathcal{S}: & \mathbb{IR}^n & \longrightarrow & \mathbb{IR}^n \times \mathbb{IR}^n \\ & [\mathbf{x}] & \longmapsto & ([\mathbf{x_{in}}], [\mathbf{x_{out}}]) \end{array}$$

with following properties:

$$\begin{array}{lrcl} \text{(i)} & [\mathbf{x}] & = & [\mathbf{x_{out}}] \cup [\mathbf{x_{in}}] \\ \text{(ii)} & [\mathbf{x_{out}}] \cap \mathbb{X} & = & [\mathbf{x}] \cap \mathbb{X} \\ \text{(iii)} & [\mathbf{x_{in}}] \cap \overline{\mathbb{X}} & = & [\mathbf{x}] \cap \overline{\mathbb{X}} \end{array}$$

We could also define the reminder $\delta\mathcal{S}$ by:

$$\delta\mathcal{S}([x]) = [\mathbf{x_{out}}] \cap [\mathbf{x_{in}}] \tag{2}$$

Explicit Definition

According to the previous definition, a separator $\mathcal{S}$ could be defined with pair of two complementary contractors $\{\mathcal{C}_{out}, \mathcal{C}_{in}\}$ such as:

$$\forall [\mathbf{x}] \in \mathbb{IR}^n,\ [\mathbf{x_{out}}] = \mathcal{C}_{out}([\mathbf{x}]) \text{ and } [\mathbf{x_{out}}] = \mathcal{C}_{in}([\mathbf{x}])$$

The reminder $\delta\mathcal{S}$ becomes:

$$\delta\mathcal{S}([\mathbf{x}]) = \mathcal{C}_{out}([\mathbf{x}]) \cap \mathcal{C}_{in}([\mathbf{x}]) \tag{3}$$

Remark: the reminder is not a separator, but a contractor on the border of the set.

Separator Algebra

The Separator algebra is a direct extension of contractor algebra. Let's take $\mathcal{S}_i = \{\mathcal{S}_i^{in}, \mathcal{S}_i^{out}\}$, We could define the following operations:

$$\begin{array}{rcll}
\mathcal{S}_1 \cap \mathcal{S}_2 & = & \left\{\mathcal{S}_1^{\text{in}} \cup \mathcal{S}_2^{\text{in}}, \mathcal{S}_1^{\text{out}} \cap \mathcal{S}_2^{\text{out}}\right\} & \text{(intersection)} \\
\mathcal{S}_1 \cup \mathcal{S}_2 & = & \left\{\mathcal{S}_1^{\text{in}} \cap \mathcal{S}_2^{\text{in}}, \mathcal{S}_1^{\text{out}} \cup \mathcal{S}_2^{\text{out}}\right\} & \text{(union)} \\
\bigcap^{\{q\}} \mathcal{S}_i & = & \left\{ \bigcap^{\{m-q-1\}} \mathcal{S}_i^{\text{in}}, \bigcap^{\{q\}} \mathcal{S}_i^{\text{out}} \right\} & \text{(relaxed intersection)} \\
\mathcal{S}_1 \backslash \mathcal{S}_2 & = & \mathcal{S}_1 \cap \overline{\mathcal{S}_2}. & \text{(difference)}
\end{array} \tag{4}$$

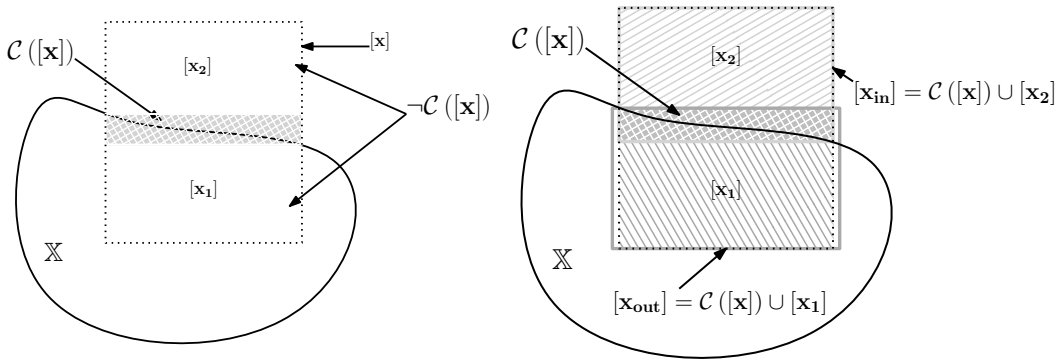
These operations are used to combine constraints. More details about separator are given in [7].

2.2.3 Border-Based Separator

When the inner and outer constraints are available, the separator is easy to build because we have an explicit form for $\mathcal{C} = \mathcal{C}_{out}$ and $\overline{\mathcal{C}} = \mathcal{C}_{in}$. But sometimes, we only have constraints which describe the border of the set $\mathbb{X}$ and they cannot be slitted into sub-constraints. So, it is not possible to isolate the inner and the outer contractor. To solve this issue, we transform the contractor into a separator. Let's take $\mathcal{C}$ as a contractor on the border of set $\mathbb{X}$. The main idea is to contract using $\mathcal{C}$ and, for each boxes in $\neg\mathcal{C}$, test if the box belongs to $\mathbb{X}$ or not. The test $\mathcal{T}$ defined by:

$$\begin{array}{rccl}
\mathcal{T}: & \mathbb{IR}^n & \longrightarrow & [0,1] \\
& [\mathbf{x}] & \longmapsto & \left\{ \begin{array}{ll} true & \text{if } [\mathbf{x}] \in \mathbb{X} \\ false & \text{otherwise} \end{array} \right.
\end{array}$$

The figure 2 illustrates the method. The box $[\mathbf{x}]$ (dotted line) is contracted on the border of set $\mathbb{X}$ using the contractor $\mathcal{C}$. Then, the two removed parts $[\mathbf{x_1}]$ and $[\mathbf{x_2}]$, which belongs to $\neg\mathcal{C}([\mathbf{x}])$, are tested with $\mathcal{T}$. For the fist one, $\mathcal{T}$ will return *true* because $[\mathbf{x_1}] \subset \mathbb{X}$, and for the other the test will return $false$. The algorithm is given by algorithm 1



(a) After contraction

(b) After identification

Figure 2: Border-Based Separator illustration

Algorithm 1: *Separator*(in: $\mathcal{C}$, $\mathcal{T}$, $[\mathbf{x_0}]$, out: $[\mathbf{x_{in}}]$,$[\mathbf{x_{out}}]$)

begin
  $[\mathbf{x}] = \mathcal{C}([\mathbf{x_0}])$;
  $[\mathbf{x_{in}}] = [\mathbf{x_{out}}] = [\mathbf{x}]$;
  $\mathbb{Z} = [\mathbf{x_0}] \backslash [\mathbf{x}]$;
  foreach *connected component* $\mathbf{y}$ *of* $\mathbb{Z}$ **do**
    if $\mathcal{T}(\mathbf{y}) == true$ **then**
      $[\mathbf{x_{in}}] = [\mathbf{x_{in}}] \cup \mathbf{y}$
    else
      $[\mathbf{x_{out}}] = [\mathbf{x_{out}}] \cup \mathbf{y}$
  return $([\mathbf{x_{in}}], [\mathbf{x_{out}}])$
end

This method will be used in the next part to build the separator used to solve our path planning problem.

2.2.4 Example of Set Inversion Using Separator

The following example illustrates how separators could be use to simplify set inversion throw a common robotics application. Let's take a robot at (x, y) measuring its distances to three landmarks with an error of ± 0.5m. The goal is to use set inversion algorithm to compute the set of all feasible locations for the robot.

Each landmark defines a set $\mathbb{X}_i$ such as :

$$\mathbb{X}_i = \left\{(x,y) \in \mathbb{R}^2,\ \sqrt{(x-x_i)^2+(y-y_i)^2} \in [d_i - 0.5, d_i + 0.5]\right\}$$

We have also set $\overline{\mathbb{X}}_i$ defined by:

$$\overline{\mathbb{X}_i} = \left\{(x,y) \in \mathbb{R}^2,\ \sqrt{(x-x_i)^2+(y-y_i)^2} \notin [d_i - 0.5, d_i + 0.5]\right\}$$

For each set we could build the associated contractor (in forward/backward way for example) $\mathcal{C}_i$ and $\overline{\mathcal{C}_i}$. If the robot's location belongs to the intersection of all $\mathbb{X}_i$ we have for the expression of the global contractor :

$$\begin{array}{rcl} \mathcal{C} & = & \mathcal{C}_1 \cap \mathcal{C}_2 \cap \mathcal{C}_3 \text{ and} \\ \overline{\mathcal{C}} & = & \overline{\mathcal{C}_1 \cap \mathcal{C}_2 \cap \mathcal{C}_3} \\ & = & \overline{\mathcal{C}}_1 \cup \overline{\mathcal{C}}_2 \cup \overline{\mathcal{C}}_3 \end{array}$$

But now, if $\mathbb{X} = (\mathbb{X}_1 \cap \mathbb{X}_3) \cup \left(\mathbb{X}_2 \cap \overline{\mathbb{X}}_3\right)$ we will have :

$$\begin{array}{rcl} \mathcal{C} & = & (\mathcal{C}_1 \cap \mathcal{C}_3) \cup \left(\mathcal{C}_2 \cap \overline{\mathcal{C}}_3\right) \\ \overline{\mathcal{C}} & = & \left(\overline{\mathcal{C}}_1 \cup \overline{\mathcal{C}}_2\right) \cap \left(\overline{\mathcal{C}}_2 \cup \mathcal{C}_3\right) \end{array}$$

So, as its was said previously, apply the *De Morgan* rules to find $\overline{\mathcal{C}}$ is not so easy and could be done automatically with the *Separator* arithmetics. When the elementary *Separator* $\mathcal{S}_i = \left\{\mathcal{C}_i, \overline{\mathcal{C}_i}\right\}$ is build the expression for the first example will be:

$$\mathcal{S} = \mathcal{S}_1 \cap \mathcal{S}_3 \cap \mathcal{S}_2$$

and for the second example :

$$\mathcal{S} = (\mathcal{S}_1 \cap \mathcal{S}_2) \cup \left(\mathcal{S}_2 \cap \overline{\mathcal{S}_3}\right)$$

Separators is an abstraction layer for contractors which allow us to manipulate combination of contractors on sets without explicitly take care about the expression of the outer contractor ($\mathcal{C}$) and the inner one ($\overline{\mathcal{C}}$).

2.3 Geometrical Constraints

This section introduces geometrical constraints, used in our problem, such as how to characterize points in a segment or in a polygon.

2.3.1 Point Inside a Curve

An elementary curve $\mathcal{C}$ is a one dimensional curve of $\mathbb{R}^2$ described by :

$$\mathcal{C} = \left\{ \mathbf{m} \in \mathbb{R}^2,\ f(\mathbf{m}) = 0 \text{ and } \mathbf{g}(\mathbf{m}) \leq 0 \right\}.$$

where f and $\mathbf{g}$ are polynomials. Complex curves are union of elementary curves. In this report, curves will be only segments.
A point $\mathbf{m}$ is inside the curve $\mathcal{C}$ if is satisfies the previous constraints.

2.3.2 Point Inside a Segment

Consider an oriented segment $[\mathbf{a}, \mathbf{b}])$ where $(\mathbf{a}, \mathbf{b}) \in \mathbb{R}^2$. The point $\mathbf{m}$ is said inside the segment if it satisfies the following constraints :

$$\begin{cases} \det(\mathbf{b} - \mathbf{a}, \mathbf{a} - \mathbf{m}) = 0 \\ \min(\mathbf{a}, \mathbf{b}) \leq \mathbf{m} \leq \max(\mathbf{a}, \mathbf{b}). \end{cases}$$

Remark The first constraint is an equality $f(x) = 0$ and the associated contractor will contract only on the border. We only have an approximation of $\mathbb{X}^+$ because there is no inner part.

2.3.3 Point Inside a Polygon

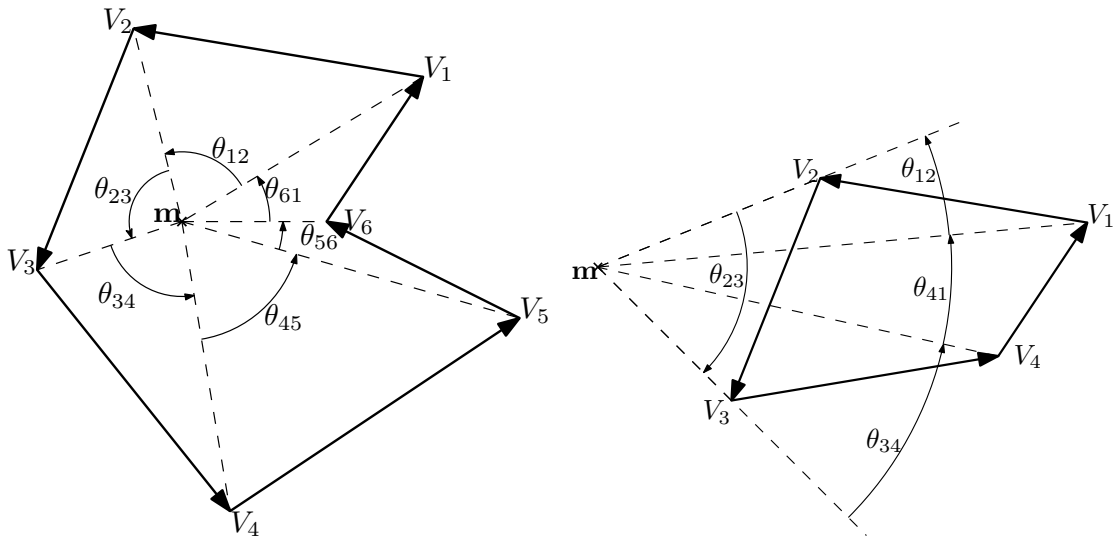
In this report, a polygon $\mathcal{P}$ is an oriented polygon, convex or not, without self interaction , composed of N segments. The border $\Delta\mathcal{P}$ of the polygon satisfies the following constraint:

$$\Delta\mathcal{P} = \left\{ \mathbf{m} \in \mathbb{R}^2,\ \exists i \in [\![1, N]\!],\ \mathbf{m} \in [\mathbf{a}_i, \mathbf{b}_i] \right\}$$

Let's us take $\mathcal{C}_{a_i,b_i}$ the contractor for the segment $[\mathbf{a_i}, \mathbf{b_i}]$, the contractor for $\Delta\mathcal{P}$ is:

$$\mathcal{C}_{\Delta\mathcal{P}} = \bigcup_{i=1}^{N} \mathcal{C}_{a_i,b_i}$$

Remark: Because the union of minimal contractor is minimal, $\mathcal{C}_{\Delta\mathcal{P}}$ is a minimal contractor for the border of the polygon P. However we have lost the information about which part is inside P and which is outside. To retrieve the different parts, we use a separator approach described in section 2.2.3.



(a) $\mathbf{wn}(\mathbf{m}, P) = 1$ (b) $\mathbf{wn}(\mathbf{m}, P) = 0$

Figure 3: In figure (a) $\mathbf{m}$ is inside P so $\mathcal{T}(\mathbf{m}) = \mathit{true}$ and in figure (b) $\mathbf{m}$ is outside P and the test is false

Separator Test

To identify if a point is inside a polygon, we use the *Winding Number* which represents the total number of times that curve travels counterclockwise around the point. The winding number depends on the orientation of the curve, and is negative if the curve travels around the point clockwise. Let's take a polygon P with vertices's $V_1, V_2, \ldots, V_n = V_1$ and $\mathbf{m}$ a point not on P the Winding Number is defined by:

$$\mathbf{wn}(\mathbf{m}, P) = \frac{1}{2\pi}\sum_{i=1}^{n}\theta_i = \frac{1}{2\pi}\sum_{i=1}^{n}\arccos\left(\frac{(V_i - \mathbf{m}).(V_{i+1} - \mathbf{m})}{\|(V_i - \mathbf{m})\|\|(V_{i+1} - \mathbf{m})\|}\right)$$

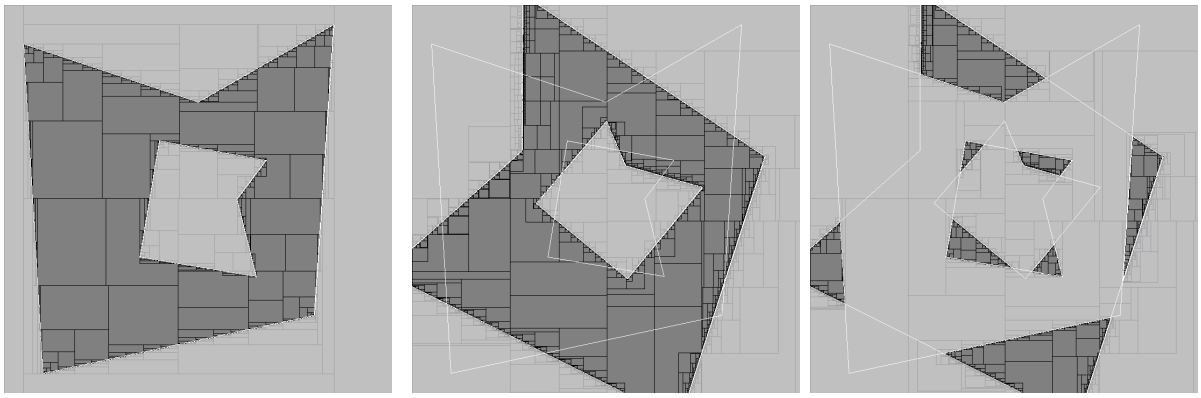
So, if $\mathbf{m}$ is outside P we will have $\mathbf{wn}(\mathbf{m}, P) = 0$, otherwise if $\mathbf{m}$ is inside, $\mathbf{wn}(\mathbf{m}, P) = 1$ (see figure 3).

Finally we build a separator $\mathcal{S}_P$ for the polygon P.The figure (4a) shows the set $\mathbb{X}$ (in dark gray) of all points inside the polygon.

2.3.4 Set Transformations

It is possible to use apply non linear transformation to a set using separator. This method is based on the Forward-Backward propagation use by separator and set inversion algorithm. The following algorithm details the steps needed for computing the expression $\mathbb{X} = f(\mathcal{S})$ where $\mathcal{S}$ is a separator set $\mathbb{Y}$ and f a function.

The following figure (4) shows some example of transformation of the set $\mathcal{S}_1$ which is the set of all points inside the polygon (in dark gray on sub-figures).



(a) Set $\mathcal{S}_1$ (b) $\mathcal{S}_2 = \mathcal{R}_{\pi/2}(\mathcal{S}_1)$ (c) $\mathcal{S}_3 = \overline{\mathcal{S}_1} \cap \mathcal{S}_2$

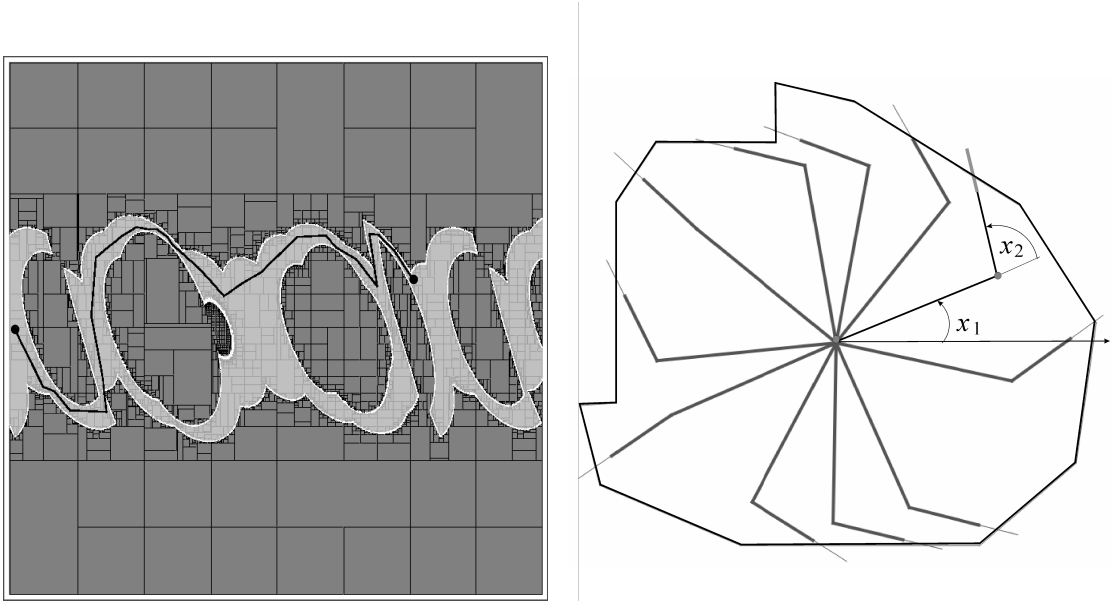
Figure 4: Figure (a) represents set $\mathcal{S}_1$ in dark gray, figure (b) shows the set $\mathcal{S}_1$ after a rotation of angle $\frac{\pi}{2}$ and the figure (c) illustrates the separator intersection. The white lines represent the border the polygon before and after the rotation

2.4 $\mathbb{C}_{free}$ inversion

As it was explained in the introduction we want to solve the following set inversion :

$$\mathbb{C}_{free} = f_1^{-1}(\mathbb{Y}) \cap f_2^{-1}\left(\overline{\mathbb{Y}}\right)$$

We have now, a separator for the Map (wire path), expressions of f_l with $l \in \{1, 2\}$ and we know how to intersect 2 separators. The figure (5) shows the result of the set inversion.



(a) Configuration space of the robot (b) Path of the robot in the world frame

Figure 5: In figure (a) : Approximation of the feasible configuration space $\mathbb{C}_{free}$ in light gray. A path corresponding to one solution of the wire loop game is depicted. Figure (b) represents the path in the world frame

The program is written in C++, and use QT for the graphic interface. Interval part is based on Luc Jaulin library As it was expected, the algorithm is able to find a feasible path to make a full rotation around the wire. It characterizes precisely all configurations which are feasible such as the wire stays inside the loop. However, in this version, the path has to be found by hand and all the space must be explored before finding a solution. Some improvements have been studied and are introduced in the next section.

3 Improvements

The previous method has the following drawbacks :

- the sub-paving could be improve by removing some boxes;
- the path has to be find by hand;
- useless part of the C-space are explored;

For each item, a solution is introduces in next sections.

3.1 Paving-reduction

In previous results, we can see boxes without any border with a yellow one. This is because the separator isn't minimal and some boxes have to be bisected before be removed by the separator. This section will introduce the concept of minimal sub-paving and will explain how to merge boxes in order to decrease the number of boxes.

3.1.1 Definition of minimal sub-paving

When the sivia algorithm is executed, an initial box is contracted and then bisected and the same thing is repeated for each part of the initial box. The resulting paving could be represented as a binary tree. Each node has either two sons or is a leaf. The i^{th} node of the tree contains the two boxes $[x_{in}](i)$ and $[x_{out}](i)$. If j is the father of the node i, we have

$$[x_{in}](i) \cap [x_{out}](i) = [x_{in}](j) \cup [x_{out}](j).$$

The binary tree is said to be minimal if for any node i_1 (not the root) with brother i_2 and father j, we have:

$$\begin{cases} (i) & [x_{in}](i_1) \neq \emptyset \text{ and } [x_{out}](i_1) \neq \emptyset \\ (ii) & [x_{in}](j) = [x_{in}](i_1) \sqcup [x_{in}](i2) \sqcup ([x](j) \setminus [x_{out}](j)) \\ (iii) & [x_{out}](j) = [x_{out}](i_1) \sqcup [x_{out}](i_2) \sqcup [x](j) \setminus [x_{in}](j) \end{cases}$$

Equation (i) means that the node is not empty. With equations (ii) and (iii) the inside and outside parts which are removed at the level node i_1 and i_2 are merged inside the node j.

3.1.2 Algorithms

From a given tree, it is possible to simplify it into an optimal tree without changing the approximation for $\mathbb{X}$. This simplification allows us to reduce the number of nodes of the tree. The algorithm is given by the listing2 which is a recursive version of the SIVIA algorithm. When a leaf is reached, the function Reduce_node (see listing 3) is called and nodes are merged.

Algorithm 2: RPaver(in: $\mathcal{S}$ inout: $[\mathbf{x}^{\text{in}}](j)$, $[\mathbf{x}^{\text{out}}](j)$, $[\mathbf{x}^{\text{in}}](i_1)$, $[\mathbf{x}^{\text{out}}](i_1)$,$[\mathbf{x}^{\text{in}}](i_2)$, $[\mathbf{x}^{\text{out}}](i_2)$

begin
- $\left\{[\mathbf{x}^{\text{in}}], [\mathbf{x}^{\text{out}}]\right\} = \mathcal{S}([\mathbf{x}^{\text{in}}](j) \cap [\mathbf{x}^{\text{out}}](j))$;
- $[\mathbf{x}] = [\mathbf{x}^{\text{in}}] \cap [\mathbf{x}^{\text{out}}]$;
- **if** $w([\mathbf{x}]) < \varepsilon$ **then**
  - **return**;
- **else**
  - Bisect $[\mathbf{x}]$ in $[\mathbf{x_1}]$, $[\mathbf{x_2}]$;
  - $[\mathbf{x}^{\text{in}}](i_1) = [\mathbf{x}^{\text{out}}](i_1) = [\mathbf{x_1}]$;
  - $[\mathbf{x}^{\text{in}}](i_2) = [\mathbf{x}^{\text{out}}](i_2) = [\mathbf{x_2}]$;
  - RPaver($\mathcal{S}$,$[\mathbf{x}^{\text{in}}](i_1)$,$[\mathbf{x}^{\text{out}}](i_1)$);
  - RPaver($\mathcal{S}$,$[\mathbf{x}^{\text{in}}](i_1)$,$[\mathbf{x}^{\text{out}}](i_1)$);
  - Tree_Reduction($\mathbf{x}(j), \mathbf{x}(i_1), \mathbf{x}(i_2)$);

end

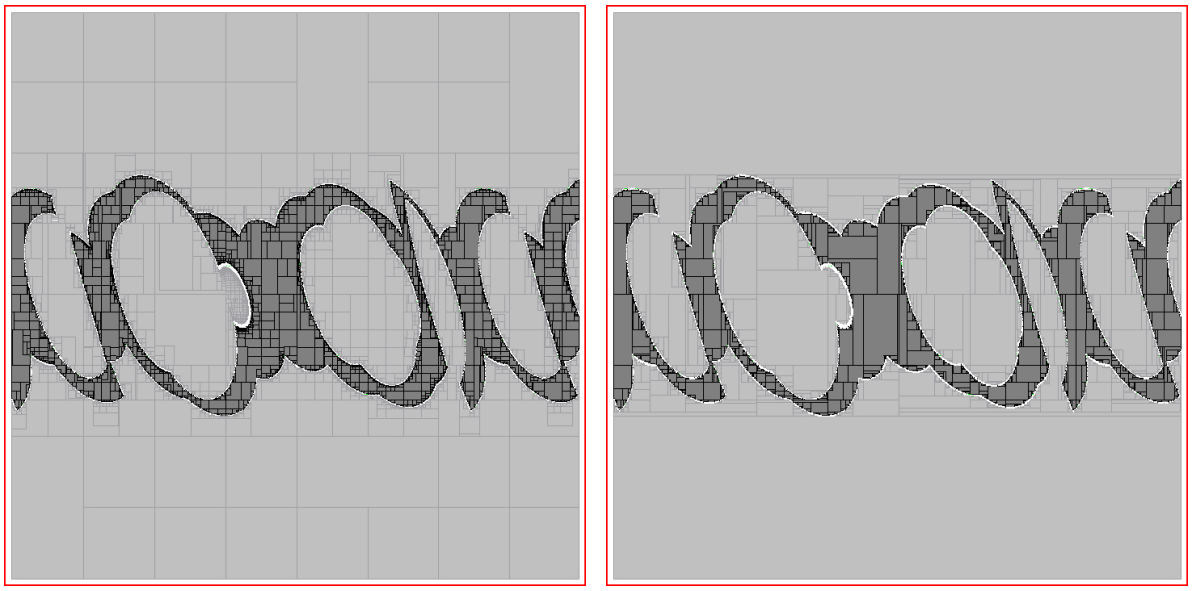
Algorithm 3: Tree_Reduction(inOut: $\mathbf{x}(j), \mathbf{x}(i_1), \mathbf{x}(i_2)$)

$[\mathbf{x}^{\text{in}}](j) = [\mathbf{x}^{\text{in}}](i_1) \sqcup [\mathbf{x}^{\text{in}}](i_2) \sqcup [\neg \mathbf{x}^{\text{out}}](j)$;
$[\mathbf{x}^{\text{out}}](j) = [\mathbf{x}^{\text{out}}](i_1) \sqcup [\mathbf{x}^{\text{out}}](i_2) \sqcup [\neg \mathbf{x}^{\text{in}}](j)$;
if $\left([\mathbf{x}^{in}](i_1) \cap [\mathbf{x}^{out}](i_1) = \emptyset \ or \ [\mathbf{x}^{in}](i_2) \cap [\mathbf{x}^{out}](i_2) = \emptyset\right)$ **then**
- remove_node_from_tree()

else
- $[\mathbf{x}] = [\mathbf{x}^{\text{in}}](j) \cap [\mathbf{x}^{\text{out}}](j)$;
- $[\mathbf{x}^{\text{in}}](i_1) = [\mathbf{x}^{\text{in}}](i_1) \cap [\mathbf{x}]$;
- $[\mathbf{x}^{\text{out}}](i_1) = [\mathbf{x}^{\text{out}}](i_1) \cap [\mathbf{x}]$;
- $[\mathbf{x}^{\text{in}}](i_2) = [\mathbf{x}^{\text{in}}](i_2) \cap [\mathbf{x}]$;
- $[\mathbf{x}^{\text{out}}](i_2) = [\mathbf{x}^{\text{out}}](i_2) \cap [\mathbf{x}]$;

3.1.3 Results

So we apply the previous algorithm on our path planning problem and as it could be seen on the figure (6) we reduce the number of boxes used to approximate $\mathbb{X}$ set. In this example, the size of the sub-paving was reduced around 30 % (33703 boxes before reduction and 22500 after). This reduction decreases only the number of boxes when the bisection's process is over, but it doesn't speed up the algorithm and decreases the computation time. At the end the sub-paving will be just easier to analyze and manipulate.



(a) Sub-paving without reduction (b) Sub-paving with reduction

Figure 6: Comparison between sub-paving with and without reduction. For example all boxes on the top have been merged

3.2 Graph representation and Cameleon algorithm

In order to find automatically a path we represent the sub-paving by an undirected graph. Each vertex represents a box and when two boxes have over-lapping face (which means that there are neighbors), an edge joints the two corresponding vertex.

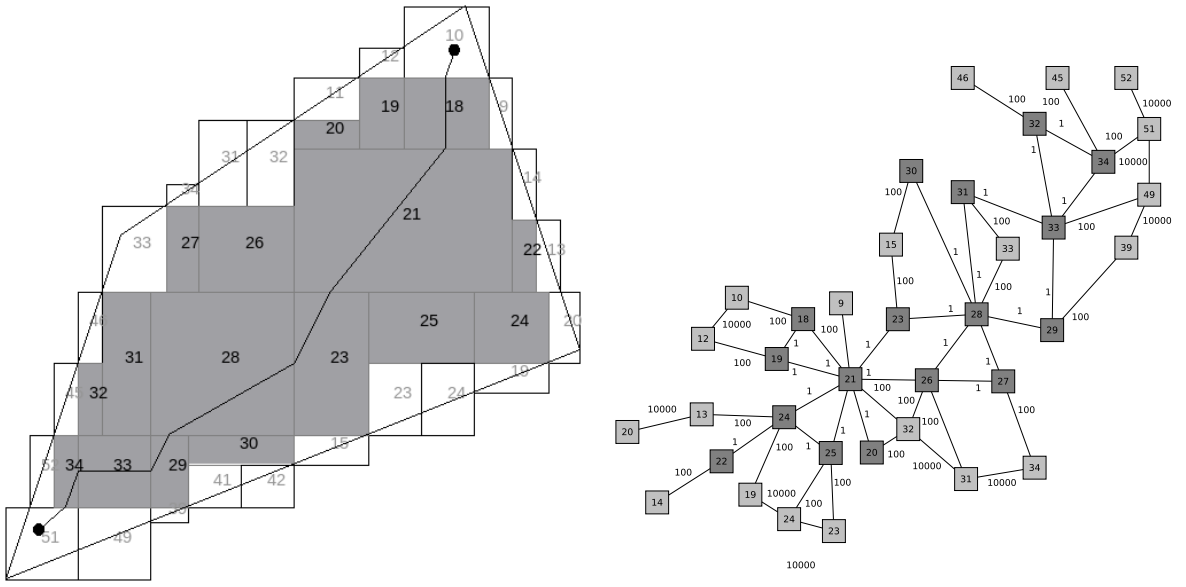
The graph is build on the fly, and while boxes are bisected and contracted, edges and vertices are added into the graph. Moreover we only add boxes which belong to $\mathbb{C}^-$ and $\mathbb{C}^+$. At the end we obtain a bi-colored undirected graph of boxes in which we could apply shortest path's algorithm such as Dijkstra's algorithm from our source configuration p_s to our goal configuration p_g. We chose to build the graph on the fly instead using the reduced one because of lake of time but it should have been more efficient.

3.2.1 Test Case

In order to test and validate the construction of the graph we use a simple polygon see composed of four edges. The corresponding graph and sub-paving are given is figure 7. As it is expected the graph is planar (see figure 7b) and the Dijkstra's algorithm works well on it. A path is found from the box number 51 to box 10. The feasible path for the wire loop game is given in figure 8.

3.2.2 Cameleon Algorithm

The previous algorithm works well and is able to find a path, when it is possible, but needs to explore all the configuration space before. A way to solve this issue was introduced by luc Jaulin [6] and is called cameleon's algorithm. The main idea is to bisect only boxes which could reach to a valid path. For this purpose, a sequence of white and gray boxes



(a) Sub-paving (b) Planar graph

Figure 7: Results of the simple test case for validating the graph construction.

are selected using Dijkstra' algorithm from the point p_s to p_g and all white boxes of it are contracted, bisected and added to the graph. Next, a new path is searched and so on until no path is found, or the sequence contains only gray boxes.

This original algorithm was modified by adding weight on edge between vertices. The main reason is that path returned by Dijkstra's algorithm is strongly binded with this weight. To speed up the we want path returned by the algorithm contains less white boxes as possible in order to limit the number of bisection. For this purpose,we use the following weight:

box's color	white	dark gray
yellow	$1e^4$	100
red	100	1

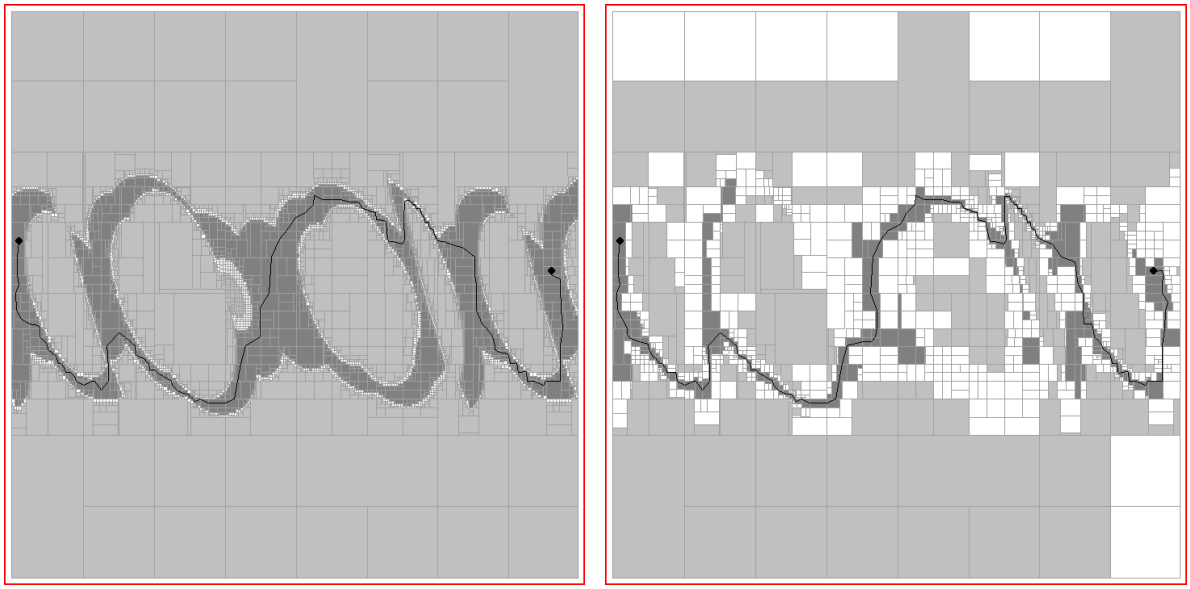
We penalize paths with white boxes and limit the passage between boxes with different colors This strategy is just an example, others strategies could be chosen such as maximize the size of boxes used.

3.2.3 Results

The figure 8 shows the results of the path finding algorithm compare to the cameleon one. Some indicators of efficiency are in table 1. As it could be seen with the wire loop game application, the cameleon is very efficient and we have 10 to 30 times less bisection than SIVIA algorithm. This number is a good indicator for the scalability of the method and has to be as low as possible. Moreover if there is no feasible path, like the figure 9, the algorithm stop quickly with the guarantee that no solution exists. So, the cameleon's algorithm using contractor seems to have good properties to solve more complex path-planning problem.

Algorithm	time (ms)	number of bisection	graph's size	epsilon
SIVIA	15490	33703	29304	0.01
SIVIA	6183	7169	6137	0.05
Cameleon	1151	1266	1438	0.01

Table 1: Comparison between the classical approach and cameleon's algorithm.



(a) Classical approach (b) Cameleon approach

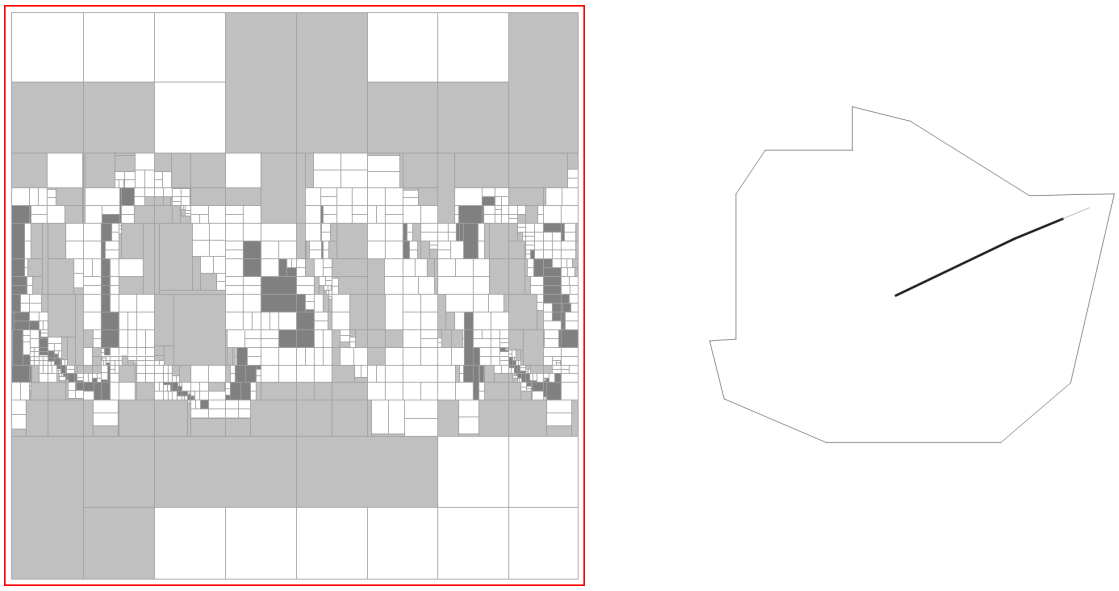
Figure 8: On (a), the sub-paving, with a feasible path, resulting from the Set inversion. On the right, same path found with camaleon's algorithm

Conclusion

Throw the wire loop game, we illustrated the possibility of doing path planning with interval analysis in an efficient and elegant way. We also created the separator object and the one associated with the constraint: "point is in a polygon" which has never been done before. Next we were able to show explicitly a feasible path throw the configuration space and implement the *Cameleon* algorithm with minor modifications.

The main advantage of interval methods seems to be their capacity to represent the $\mathbb{C}$-space as tree or graph composed of continuous boxes without loosing any solution, and on which we could apply very efficient graph algorithms. These methods are guaranteed and are able to prove that no path exist. To my knowledge, it is a key point which could distinguish interval methods from the others.

Future work should be to compare with different existing methods and also try others classical path planning problems.



(a) Configuration space (b) World frame space

Figure 9: No path was found by the algorithm in (a) because the robot can't keep his loop around the wire. The configuration is shown on the figure (b)

References

[1] F. Abdallah, A. Gning, and P. Bonnifait. Box particle filtering for nonlinear state estimation using interval analysis. *Automatica*, 44(3):807–815, 2008.

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

[3] R. Daily and D.M. Bevly. Harmonic potential field path planning for high speed vehicles. In *American Control Conference, 2008*, pages 4609–4614, 2008.

[4] 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.

[5] A. Gning and P. Bonnifait. Constraints propagation techniques on intervals for a guaranteed localization using redundant data. *Automatica*, 42(7):1167–1175, 2006.

[6] L. Jaulin. Path planning using intervals and graphs. *Reliable Computing*, 7(1):1–15, 2001.

[7] L. Jaulin and B. Desrochers. Separators: a new interval tool to bracket solution sets; application to path planning. *Submitted to Engineering Applications of Artificial Intelligence*, 2014.

[8] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500–505, 1985.

[10] 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.

[11] S. Lagrange, N. Delanoue, and L. Jaulin. Injectivity analysis using interval analysis. application to structural identifiability. *Automatica*, 44(11):2959–2962, 2008.

[12] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[13] O. Lévêque, L. Jaulin, D. Meizel, and E. Walter. Vehicule localization from inaccurate telemetric data: a set inversion approach. In *Proceedings of 5th IFAC Symposium on Robot Control SY.RO.CO.'97*, volume 1, pages 179–186, Nantes, France, 1997.

[14] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *"IEEE Transactions on Systems Man and Cybernetics"*, 11(10):681–698, 1981.

[15] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

[16] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[17] R. Pepy, M Kieffer, and E Walter. Reliable robust path planning. *International Journal of Applied Mathematics and Computer Science*, 3(19):413–424, 2009.

[18] N. Ramdani and P.Poignet. Robust dynamic experimental identification of robots with set membership uncertainty. *IEEE/ASME Transactions on Mechatronics*, 10(2):253–256, 2005.