MOTION PLANNING USING INTERVAL ANALYSIS

Luc JAULIN and **Alain GODON**

Laboratoire d'Ingénierie des Systèmes Automatisés,
Faculté des Sciences, 2 Boulevard Lavoisier,
49 045 Angers Cédex, France
http://www.istia.univ-angers.fr/~jaulin/
http://www.istia.univ-angers.fr/~godon/
{jaulin,godon}@univ-angers.fr

Abstract: This paper deals with characterizing the connected component $\mathcal{A}_{\mathcal{S}}(\vec{a})$ of a set $\mathcal{S}$ that contains a given point $\vec{a}$. The set $\mathcal{S}$ is assumed to be defined by nonlinear inequalities, so that interval analysis can be used to prove that a given box is inside or outside $\mathcal{S}$. Further, the characterization of $\mathcal{A}_{\mathcal{S}}(\vec{a})$ is used to find a feasible path (*i.e.* a path that lies within $\mathcal{S}$) that links $\vec{a}$ to any other point $\vec{b} \in \mathcal{A}_{\mathcal{S}}(\vec{a})$.

Keywords: Connexity, Interval analysis, Motion Planning, Set inversion, Subpavings.

1. Introduction

In this paper, we present a new interval-based approach to find a collision-free path for a robot in a given space with obstacles. The issue of path planning in a known environment has been addressed by many researchers (see *e.g.* [7], [8] and [4]). Most of the current approaches to path planning are based on the concept of configuration space (C-space) introduced by Lozano-Pérez and Wesley [5]. C-space is the set $\mathcal{S}$ of all possible configurations of a robot. The start configuration and the goal configuration become two points $\vec{a}$ and $\vec{b}$ of $\mathcal{S}$. The number of independent parameters needed to fully specify a robot configuration is the dimension of the C-space.

The problem formulated in the C-space amounts to find a path included in $\mathcal{S}$ that links $\vec{a}$ to $\vec{b}$. Many approaches to solve this problem are based on the use of potential functions, first introduced by Khatib [3]. In the potential field approach, the obstacles to be avoided are represented by a repulsive potential, and the goal is represented by an attractive potential. According to the force generated by the sum of these potential fields robot attains (if the method does not stop at any local minimum) the goal without collding with obstacles. Except for some special classes of problems, these methods are not guaranteed and interval analysis could be helpful to deal with motion planning problems in a global and guaranteed way. Interval analysis has already been used for parametric paths in [2], but the former method requires a model for the path and is limited to small dimensional path models.

In this paper a new nonparametric approach based on interval analysis is considered. This approach first characterizes the set $\mathcal{S}$ of all feasible configurations for the robot as presented on Section 3. Then, a color propagation algorithm, presented in Section 4, is performed to characterize the connected component $\mathcal{A}_{\mathcal{S}}(\vec{a})$ of $\mathcal{S}$ that contains the start configuration point $\vec{a}$. At last, a feasible path that links $\vec{a}$ to $\vec{b}$ is computed using the procedure presented on Section 5.

2. Inclusion test

Interval arithmetic provides an extension to real intervals and vector intervals of the classical arithmetical operations on real numbers and vectors. (see e.g. [6]). It makes it possible to build inclusion tests to prove that a given box $\vec{X}$ is inside or outside the configuration space $\mathcal{S}$, when $\mathcal{S}$ is given by nonlinear inequalities.

Box: A *box* or *vector interval* $\vec{X}$ of $\mathbb{R}^n$ is the Cartesian product of n intervals:

$$\vec{X} = \left[x_1^-, x_1^+\right] \times \cdots \times \left[x_n^-, x_n^+\right]. \tag{2.1}$$

The set of all boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$. To *bisect* a box $\vec{X}$ means to cut it along the symmetry plane normal to the side of maximum length. The length of this side is the width of $\vec{X}$ and denoted by $w(\vec{X})$. A bisection of $\vec{X}$ generates two non-overlapping boxes $\vec{X}_L$ and $\vec{X}_R$ such that $\vec{X} = \vec{X}_L \cup \vec{X}_R$. A box $\vec{Y}$ is a *left neighbor* of $\vec{X}$ with respect to the ith dimension if and only if

$$y_i^+ = x_i^- \text{ and } \forall j \in [1, n],\ X_j \cap Y_j \neq \emptyset \tag{2.2}$$

We shall also say that $\vec{Y}$ is a i-left neighbor of $\vec{X}$. $\vec{Y}$ is a *i-right neighbor* of $\vec{X}$ if and only if $\vec{X}$ is a i-left neighbor of $\vec{Y}$.

Boolean Interval: A Boolean number is an element of $\mathbb{B} = \{0, 1\}$ where 0 stands for false and 1 for true. By extension, a Boolean interval is an element of $\mathbb{IB} = \{0, 1, [0, 1]\}$. If a is a Boolean interval,

$$0.a = 0;\ 1.a = a;\ 0 + a = a;\ 1 + a = 1;\ a.a = a + a = a. \tag{2.3}$$

Inclusion test: An *inclusion test* for the Boolean function (or *test*) $t : \mathbb{R}^n \rightarrow \{0, 1\}$ is a function $T : \mathbb{IR}^n \rightarrow \mathbb{IB}$ such that for all boxes $\vec{X} \in \mathbb{IR}^n$,

$$\begin{array}{rcl} T(\vec{X}) = 1 & \Rightarrow & \forall \vec{x} \in \vec{X},\ t(\vec{x}) = 1, \\ T(\vec{X}) = 0 & \Rightarrow & \forall \vec{x} \in \vec{X},\ t(\vec{x}) = 0. \end{array} \tag{2.4}$$

Moreover, for all vectors $\vec{a}$, $T(\vec{a}) = t(\vec{a})$. Interval analysis provides inclusion tests for a large class of Boolean tests.

3. Characterization of the configuration space

3.1. Paving

A guaranteed characterization of compact sets of $\mathbb{R}^n$ can be done by using pavings. A *paving* of a box $\vec{X}_0$ is partition of $\vec{X}_0$ with boxes. On Figure 1, a paving of the box $\vec{X}_0 = [0, 8] \times [0, 6]$ is presented. This paving contains 3 boxes $\vec{A}, \vec{B}$ and $\vec{X}$.

In the algorithm to be presented, topological informations on boxes are required. Each box has to memorize the address of its neighbors. Before defining how is implemented the box class, let us have a look to the Figure 2. It illustrates how these topological informations take place in the memory of the computer. Boxes $\vec{X}, \vec{A}$ and $\vec{B}$ are rounded by four grey trapezoid, each of them representing a list of neighbors. For example, since $\vec{A}$ has two right neighbors in the direction 1, two arrows have their origins located on the right trapezoid of $\vec{A}$.

Figure 1: A paving with 3 boxes

Figure 2: Representation in memory of the paving drawn on Figure 1

C++ notations. In this paper, a C++-based notation is used. Recall that if A is an object (for example an interval), $\&A$ denotes the address of A. A *pointer* is an address. If p is a pointer, $*p$ denotes the object located at the address p. We say that the pointer p points toward the object $*p$. For example, if $p = \&A$, $*p = A$. An object A contains fields. For example, if the object A belongs to the interval class, it contains two fields: *inf* and *sup*. If p points toward A, and $*p = A = [3, 4]$, then, $*p.inf = A.inf = p \rightarrow inf = 3$. Usually, the notation $p \rightarrow inf$ is preferred on $*p.inf$. A declaration of an object A which belongs to the box class is: "box A;", a declaration of a pointer to a box is "box* p;". The declaration of an array *tab* of n intervals is "interval[n] *tab*".

The box class to be used is defined as follows:

```
class Box  {  interval[n]              val;
              {0, 1, [0, 1]}           in;
              {grey, white, black}     color;
              List[n]                  left;
              List[n]                  right;
              Box*                     previous;  }
```

Val: $\vec{X}.val[i]$ is the ith component interval of $\vec{X}$. For simplicity of notations, $\vec{X}.val[i]$ will often be denoted by X_i.

In: $\mathcal{A}$.in= 0 (resp. $1, [0,1]$), means that $\vec{X} \cap \mathcal{S} = \emptyset$ (resp. $\vec{X} \subset \mathcal{S}$, $\vec{X}$ is ambiguous).

Color: This field will be used to implement the color-propagation algorithm PAINT given on Subsection 4.3.

Left: In the paving, for a given direction i, all left neighbors of $\vec{X}$ are referenced in the lists left[i]. Recall that in C++, the declaration "List[n] *left*", creates an array *left* of n lists. Each list of this array contains pointers toward corresponding boxes. For example, on the paving represented on Figure 2, $\vec{B}.left[1] = \{\&\vec{A}\}$, *i.e.* the list $\vec{B}.left[1]$ only contains the address $\&\vec{A}$ of $\vec{A}$. Note that since $\vec{B}$ has no neighbors below itself, $\vec{B}.left[2] = \emptyset$.

Right: The list $\vec{X}$.right[i] contains pointers to boxes that are on the right neighborhood of $\vec{X}$ (with respect to the ith dimension). For instance, on the paving of Figure 2, $\vec{A}.right[1] = \{\&\vec{X}, \&\vec{B}\}$.

Previous: $\vec{X}$.previous is a pointer toward the box responsible for its color. This field will be used to find an admissible path on Section 5.

3.2. Bisection Algorithm

When a box $\vec{X}$ of the paving is bisected, it is replaced by two boxes $\vec{X}_L$ and $\vec{X}_R$. A box $\vec{N}$ that was a neighbor of $\vec{X}$ may also be neighbor of $\vec{X}_L$ or/and $\vec{X}_R$. The following algorithm bisects $\vec{X}$ updating all right and left fields of the paving.

Bisect (box $\vec{X}$, box $\vec{X}_L$, box $\vec{X}_R$)

1 i = principal axe ($\vec{X}$);
2 Copy $\vec{X}$ into $\vec{X}_L$ and into $\vec{X}_R$;
3 $\vec{X}_L[i] = [x_i^-; \frac{x_i^- + x_i^+}{2}]$; $\vec{X}_R[i] = [\frac{x_i^- + x_i^+}{2}; x_i^+]$;
4 $\vec{X}_L.right[i] = \{\&\vec{X}_R\}$ and $\vec{X}_R.left[i] = \{\&\vec{X}_L\}$
5 For all $\vec{N}$ referenced in $\vec{X}.left[i]$, replace $\&\vec{X}$ by $\&\vec{X}_L$ in the list $\vec{N}.right[i]$;
6 For all $\vec{N}$ referenced in $\vec{X}.right[i]$, replace $\&\vec{X}$ by $\&\vec{X}_R$ in the list $\vec{N}.left[i]$;
7 For all $k \in [1, n]$, $k \neq i$,
8 For all $\vec{N}$ referenced in $\vec{X}.left[k]$,
9 Remove $\&\vec{X}$ from the list $\vec{N}.right[k]$;
10 If $\vec{X}_L$ is a k-right neighbor of $\vec{N}$, add $\&\vec{X}_L$ to $\vec{N}.right[k]$
11 else remove $\&\vec{N}$ from $\vec{X}_L.left[k]$;
12 If $\vec{X}_R$ is a k-right neighbor of $\vec{N}$, add $\&\vec{X}_R$ to $\vec{N}.right[k]$
13 else remove $\&\vec{N}$ from $\vec{X}_R.left[k]$;
14 For all $\vec{N}$ referenced in $\vec{X}.right[k]$,
15 Remove $\&\vec{X}$ from the list $\vec{N}.left[k]$;
16 If $\vec{X}_L$ is a k-left neighbor of $\vec{N}$, add $\&\vec{X}_L$ to $\vec{N}.left[k]$
17 else remove $\&\vec{N}$ from $\vec{X}_L.right[k]$;
18 If $\vec{X}_R$ is a k-left neighbor of $\vec{N}$, add $\&\vec{X}_R$ to $\vec{N}.left[k]$
19 else remove $\&\vec{N}$ from $\vec{X}_R.right[k]$;
20 Next k;

Table 1: Bisection of a box

Figure 3 gives a representation of the new paving obtained when the box $\vec{X}$ has been bisected.

Figure 3: Representation of the paving obtained after a bisection of the box $\vec{X}$ of Figure 2

Comments: In order to explain the bisection algorithm, let us consider the paving of Figure 2 where the box $\vec{X}$ has to be bisected. Since $\vec{X} = [4,8] \times [3,6]$, at Step 1, we get $i = 1$. After Step 3, $\vec{X}_L = [4,6] \times [3,6]$, $\vec{X}_R = [6,8] \times [3,6]$, moreover, since a full copy has been performed at Step 2, $\vec{X}.left[1] = \vec{X}_L.left[1] = \vec{X}_R.left[1] = \{\&\vec{A}\}$, $\vec{X}.left[2] = \vec{X}_L.left[2] = \vec{X}_R.left[2] = \{\&\vec{B}\}, \ldots$. After Step 4, $\vec{X}_L.right[1] = \{\&\vec{X}_R\}$ and $\vec{X}_R.left[1] = \{\&\vec{X}_L\}$. Before Step 5, $\vec{A}.right[1] = \{\&\vec{X}, \&\vec{B}\}$, after $\vec{A}.right[1] = \{\&\vec{X}_L, \&\vec{B}\}$. Since $\vec{X}.right[1] = \emptyset$, Step 6 is not executed. Because $n = 2$ and $i = 1$, the *for* loop between Steps 7 to 20, is run only once with $k = 2$. At Step 8, $\vec{X}.left[2] = \{\&\vec{B}\}$, the *for*-loop is then performed only once with $\vec{N} = \vec{B}$. Before Step 9, $\vec{B}.right[2] = \{\&\vec{X}\}$, after $\vec{B}.right[2] = \emptyset$. At Step 10, since $\vec{X}_L$ is a 2-right neighbor of $\vec{B}$, $\&\vec{X}_L$ is inserted in the empty list $\vec{B}.right[2]$. The condition of Step 12 is also satisfied and then, after Step 12, $\vec{B}.right[2] = \{\&\vec{X}_L, \&\vec{X}_R\}$. Because $\vec{X}.right[2] = \emptyset$, Steps 14 to 19 are skipped. We then obtain the configuration illustrated by Figure 3. $\diamondsuit$

3.3. Set Inversion Algorithm

It is assumed that *(i)* the configuration space $\mathcal{S}$ is a compact set of $\mathbb{R}^n$ included in a possibly very large box $\vec{X}_0$, *(ii)* the start configuration point $\vec{a}$ belongs to $\mathcal{S}$ and *(iii)* an inclusion test $T(\vec{X})$ for the test $t(\vec{x}) \Leftrightarrow (\vec{x} \in \mathcal{S})$ is available. The algorithm given on Table 2 characterizes $\mathcal{S}$. This algorithm SIVIA (for Set Inversion Via Interval Analysis) has already been presented in [1] but is adapted to keep track of topological informations on boxes. Moreover, it returns a pointer p_a to the box that contains $\vec{a}$.

Comments: The list $\mathcal{L}$ contains pointers to boxes that have not yet been proved to be inside or outside $\mathcal{S}$. After Step 3, $\mathcal{L} = \{\&\vec{X}_0\}$ where $\&\vec{X}_0$ is the address of $\vec{X}_0$. At Step 5, a box pointer is removed from $\mathcal{L}$ and is stored in p. At Step 6, $T(*p)$, where $*p$ is the box pointed by p, is stored in $p \rightarrow in$. The bisection routine is performed at Steps 7, 8 and 9. If the feasibility of the current box $*p$ is unknown and if $*p$ is big enough to be bisected, two new boxes resulting from the bisection of $*p$ are created and their addresses are stored in $\mathcal{L}$. The box $*p$, now replaced by its two subboxes, is deleted from the memory of the computer. The pointer p still exists ($p = NULL$) but $*p$ does not exist anymore. A box for which the condition of Step 7 is not satisfied will belong to the final paving and should not to be deleted. The condition of Step 10 is evaluated in this case. If the current box contains $\vec{a}$, the address of this box is stored in the pointer p_a. This pointer will be used for the initialization of the color propagation routine

PAINT presented on the next section. At Step 11, when $\mathcal{L}$ is empty, a paving of $\vec{X}_0$ exists in the memory. All boxes of this subpaving can now be reached from p_a. ♢

SIVIA(Vector $\vec{a}$, Box $\vec{X}_0$, Inclusion test T)

```
1   If T(a) ≠ 1, return ("Error : a should belong to S");
2   L = ∅;
3   Store &X0 into the list L;
4   Do
5       Take any box pointer p of L;
6       p → in = T(*p);
7       If (p → in == [0,1] and w(*p) > ε)
8           Bisect *p and store the pointers of the two resulting boxes in L;
9           Delete *p;
10      Else if a ∈*p, p_a = p;
11  While (L ≠ ∅);
```

Table 2: Algorithm for enclosing the configuration space

4. Characterization of the reachable set

4.1. Reachable set

Let $\mathcal{S}$ be a set of $\mathbb{R}^n$. Two points $\vec{a}$ and $\vec{b}$ of $\mathcal{S}$ are $\mathcal{S}$-connected if there exists a *path* from $\vec{a}$ to $\vec{b}$ that lies within $\mathcal{S}$. The *reachable set* $\mathcal{A}_\mathcal{S}(\vec{a})$ is the set of all $\vec{x}$ such that $\vec{a}$ and $\vec{x}$ are $\mathcal{S}$-connected and can be interpreted as the connected component of $\mathcal{S}$ that contains $\vec{a}$. Among all properties satisfied by $\mathcal{A}_\mathcal{S}(\vec{a})$ let us quote :

$$
\begin{array}{lllll}
\text{(i)} & \mathcal{A}_\mathcal{S}(\vec{a}) \subset \mathcal{S} & & & \\
\text{(ii)} & \mathcal{S} \text{ is connected,} & \Rightarrow & \mathcal{A}_\mathcal{S}(\vec{a}) = \mathcal{S} & \\
\text{(iii)} & \vec{b} \in \mathcal{A}_\mathcal{S}(\vec{a}), & \Rightarrow & \vec{a} \in \mathcal{A}_\mathcal{S}(\vec{b}) & \\
\text{(iv)} & \vec{b} \in \mathcal{A}_\mathcal{S}(\vec{a}), & \Rightarrow & \mathcal{A}_\mathcal{S}(\vec{a}) = \mathcal{A}_\mathcal{S}(\vec{b}) & \\
\text{(v)} & \mathcal{A}_\mathcal{S}(\vec{a}) \cap \mathcal{A}_\mathcal{S}(\vec{b}) \neq \emptyset & \Leftrightarrow & \mathcal{A}_\mathcal{S}(\vec{a}) = \mathcal{A}_\mathcal{S}(\vec{b}) &
\end{array}
$$

The problem to be solved is the characterization of the reachable set $\mathcal{A}_\mathcal{S}(\vec{a})$. The color-propagation algorithm PAINT presented on Subsections 4.3 and 4.2 allows a guaranteed characterization of $\mathcal{A}_\mathcal{S}(\vec{a})$. It is based on the three following properties.

$$
\begin{array}{lllll}
\text{(i)} & \vec{a} \in \vec{X} \text{ and } \vec{X} \subset \mathcal{S} & \Rightarrow & \vec{X} \subset \mathcal{A}_\mathcal{S}(\vec{a}) & \\
\text{(ii)} & \vec{X}_1 \cap \vec{X} \neq \emptyset \text{ and } \vec{X}_1 \subset \mathcal{A}_\mathcal{S}(\vec{a}) \text{ and } \vec{X} \subset \mathcal{S} & \Rightarrow & \vec{X} \subset \mathcal{A}_\mathcal{S}(\vec{a}) & (4.1) \\
\text{(iii)} & \vec{X} \cap \mathcal{S} = \emptyset & \Rightarrow & \vec{X} \cap \mathcal{A}_\mathcal{S}(\vec{a}) = \emptyset &
\end{array}
$$

where $\vec{X}$ and $\vec{X}_1$ are two given boxes.

4.2. Enclosure of the reachable set

The following algorithm, based on the three properties (4.1), builds two subpavings $\mathcal{A}^-$ and $\Delta\mathcal{A}$ such that $\mathcal{A}^- \subset \mathcal{A}_\mathcal{S}(\vec{a}) \subset \mathcal{A}^- \cup \Delta\mathcal{A}$.

1. By using the algorithm SIVIA presented on Section 3, partition $\vec{X}_0$ with boxes: those that have been proved to be inside $\mathcal{S}$, those that have been proved to be outside $\mathcal{S}$ and those whose width is smaller that ε.

2. Paint black the box $\vec{X}_a$ that contains $\vec{a}$. Paint black all neighbors $\vec{X}$ of a black box that satisfy $\vec{X}.in \neq 0$.

3. If $\vec{X}_a.in = 1$, paint grey $\vec{X}_a$. Paint grey all neighbors $\vec{X}$ of a grey box that satisfy $\vec{X}.in = 1$.

4. Store all grey boxes into $\mathcal{A}^-$ and all black boxes into $\Delta\mathcal{A}$.

Steps 2 and 3 of this algorithm are performed by calling the procedure PAINT, presented on the next subsection, as follows:

```
2  PAINT(p_a, {[0,1],1}, black);
3  PAINT(p_a, {1}, grey);
```

4.3. The color-propagation Algorithm PAINT

The routine PAINT is a recursive routine first call with $p_1 = p_a$. PAINT colors the box $*p_1$ and calls itself for the neighbors of $*p_1$ that satisfy the right condition.

```
PAINT(box* p_1, IntervalBooleanList B, color c)
1  For i = 1 to n
2        p_1 → color = c;
3        For i = 1 to n
4              For all p in p_1 → left[i] and all p in p_1 → right[i]
5                    if (p → color ≠ c) and (p → in ∈ B), PAINT(p, B, c);
6  Next i;
```

Table 3: The recursive color propagation algorithm

Comments: The type *IntervalBooleanList* represents a set of Boolean intervals. In our context this set is either $\mathcal{B} = \{[0,1],1\}$ or $\mathcal{B} = \{1\}$. Only boxes whose *in* field belongs to $\mathcal{B}$ and that are not painted yet are allowed to be painted. ♢

4.4. Test case

Let us consider the set

$$\mathcal{S} = \{(x,y) |\, (t_1(x,y) \text{ or } t_2(x,y)) \text{ and } t_3(x,y)\} \tag{4.2}$$

where the tests t_1, t_2 and t_3 are defined by

$$\begin{array}{rcrcc} t_1(x,y) & \Longleftrightarrow & \sin(\sqrt{x^2+y^2}) & \in & [0.5;1] \\ t_2(x,y) & \Longleftrightarrow & \sin(0.2x^2).\sin(0.2y^2).\sin(0.3\left(x^2+y^2\right)) & \in & [0.3;0.5] \\ t_3(x,y) & \Longleftrightarrow & \sqrt{x^2+y^2} & \in & [0,20] \end{array} \tag{4.3}$$

With $\vec{a} = (-14,0)$ and $\vec{b} = (14,0)$, the paving generated by our algorithm is represented on Figure 4 for $\varepsilon = 0.4$ and on Figure 5 for $\varepsilon = 0.2$. Four zones partition the plane:

1. the grey zone represents the set $\mathcal{A}^-$ of all points that are proved to be inside $\mathcal{A}_{\mathcal{S}}(\vec{a})$

2. the white zone represents the points that are proved to be outside $\mathcal{S}$

3. the zone with a brick texture represents the points that are proved to be inside $\mathcal{S}$, but outside $\mathcal{A}_{\mathcal{S}}(\vec{a})$

4. and the black zone repre
is not proved.

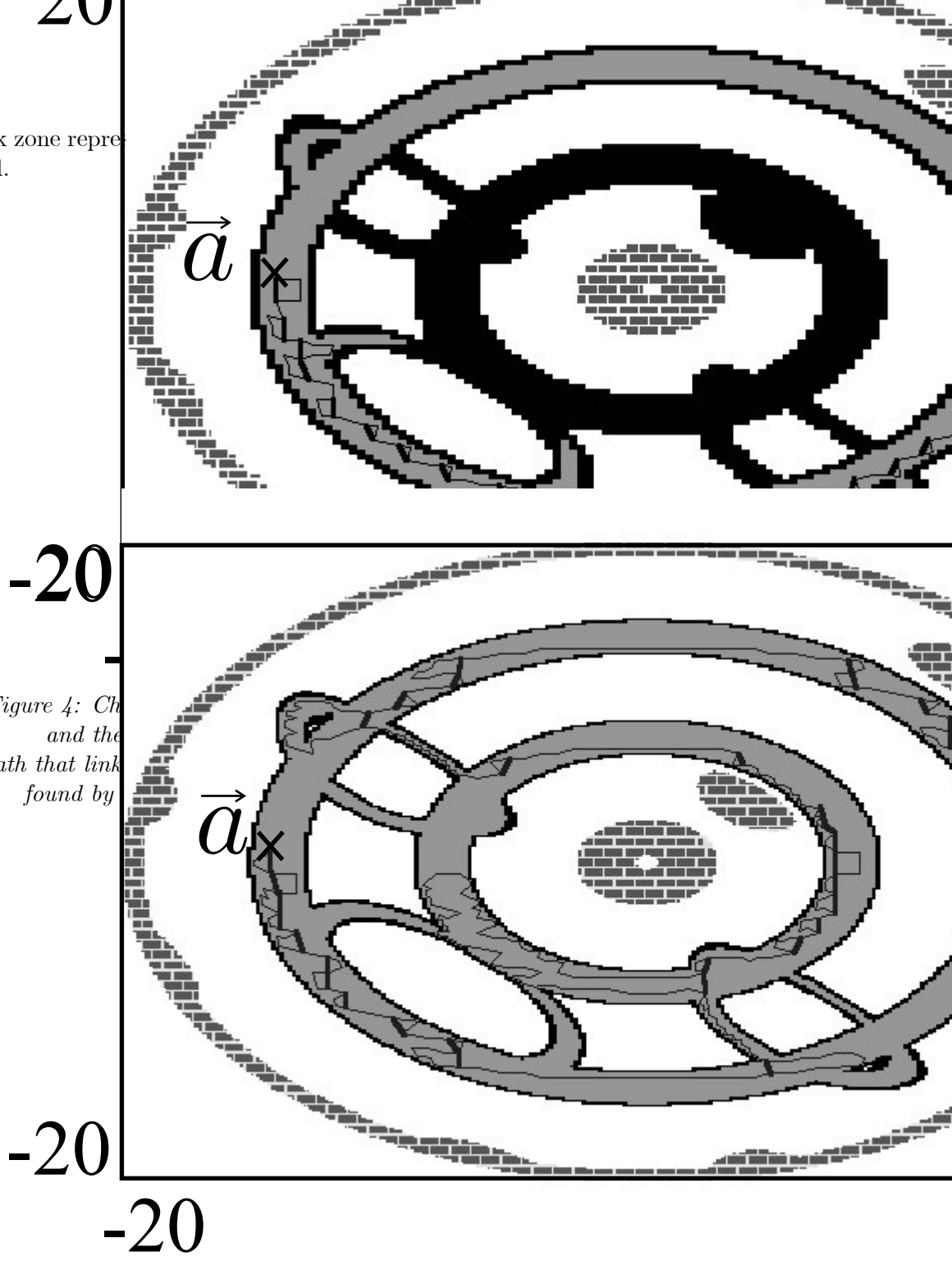

Figure 4: Ch
and the
The basic path that link
found by

Figure 5: Characterization of sets $\mathcal{S}$ and $\mathcal{A}_{\mathcal{S}}(\vec{a})$ for $\varepsilon = 0.2$.
The paths that are obtained are much more chaotic than for $\varepsilon = 0.4$.

5. Feasible Path

5.1. Basic path

Assume that a non empty inner set $\mathcal{A}^-$ for $\mathcal{A}_{\mathcal{S}}(\vec{a})$ has been generated. Consider a point $\vec{b}$ in $\mathcal{A}^-$. Since $\mathcal{A}_{\mathcal{S}}(\vec{a})$ is non empty, $p_a \rightarrow in = 1$. Since $\vec{b} \in \mathcal{A}^-$, a pointer to the box of the paving that contains $\vec{b}$ can be found by inserting at the beginning of the algorithm PAINT the statement

$$\text{if } \vec{b} \in p_1 \text{ then } p_b = p_1;$$

Since both $\vec{a}$ and $\vec{b}$ belong to $\mathcal{A}^- \subset \mathcal{A}_{\mathcal{S}}(\vec{a})$, there exists a feasible path from $\vec{a}$ to $\vec{b}$. The *previous* field of the box class can be used to find a feasible path. Let us insert in the body of the *if* statement at Step 5 of the PAINT algorithm:

$$p \rightarrow previous = p_1;$$

The algorithm of Table 4 generates a list $\mathcal{L}$ of feasible points. The first point of $\mathcal{L}$ is $\vec{a}$ and the last one is $\vec{b}$.

$\boxed{\mathcal{L} = \text{BASICPATH}(\text{box* } p)}$

1 If $p == NULL$ return $(\{\vec{a}\})$
2 If $\vec{b} \in {}^*p$, return (BASICPATH$(p_b \rightarrow previous) \cup \{\text{center}({}^*p), \vec{b}\}$)
3 else return BASICPATH$(p \rightarrow previous) \cup \{\text{center}({}^*p)\}$

Table 4: An algorithm to find a feasible path from $\vec{a}$ to $\vec{b}$

Comments: At Step 1, if p points nowhere, the algorithm is at the end of the recursive chain. The point $\vec{a}$ is thus inserted as the first element of $\mathcal{L}$. If the condition of Step 2 is satisfied, BASICPATH is called for the first time with $p = p_b$. $\vec{b}$ is inserted at the end of $\mathcal{L}$ and center$({}^*p_b)$ just before $\vec{b}$. For intermediate points, Step 3 is performed and the center of the current box is inserted in $\mathcal{L}$. ♢

The points of $\mathcal{L}$, linked with segments yields the *basic path*. Since, the bisection strategy of SIVIA is based on the principal plane, for two adjacent boxes $\vec{X}$ and $\vec{Y}$ of the generated paving, the segment [center$(\vec{X})$, center$(\vec{Y})$] is a subset of $\vec{X} \cup \vec{Y}$. The basic path is therefore feasible, *i.e.* the basic path is a subset of $\mathcal{S}$. For the test case presented in subsection 4.4, the basic path obtained for $\varepsilon = 0.4$ and the one obtained for $\varepsilon = 0.2$ are drawn on Figure 4 and 5 with thin lines.

5.2. Improved path

As shown on the test case, the basic path is very chaotic. Although many methods could be developed to find a good feasible path, we shall propose here a procedure IMPROVEPATH to improve the path generated by BASICPATH. The principle of IMPROVEPATH is to remove some unnecessary loops. Figure 6 presents a path assumed to be generated by BASICPATH. The arrows represent the *previous* field of the box class. Boxes are numbered from $\vec{b}$ to $\vec{a}$. The principle of IMPROVEPATH is to scan the basic path from $\vec{b}$ to $\vec{a}$. When possible, it shortcuts the path. For example, when IMPROVEPATH reaches the box 9, it goes to box 13. When it reaches the box 20, it does not try to shortcut via Step 11 because 21 is bigger than 11. Following this strategy, the improved path is represented on Figure 7.

Figure 6: A typical path that could be generated by BasicPath

Figure 7: The improved path associated with the basic path of Figure 6.

To implement this algorithm, the field *number* is added to the box class. The procedure BasicPath is adapted to number each box of the path. The constructor of the box class sets the field number to 0 for any new box. Once BasicPath has been performed, the procedure ImprovePath(p_b) presented on Table 5 is called.

$\boxed{\mathcal{L} = \text{ImprovePath(box* } p)}$

1 If $p == NULL$ return $(\{\vec{a}\})$
2 If $\vec{b} \in {*p}$, return (ImprovePath$(p_b \rightarrow previous) \cup \{\text{center}(*p), \vec{b}\}$)
3 For all neighbors $*pY$ of $*p$
4 If $pY \rightarrow number > p \rightarrow number + 1$,
5 $p \rightarrow previous = pY$; Exit from the For loop;
6 return ImprovePath$(p- > previous) \cup \{\text{center}(*p)\}$

Table 5: Procedure to improve the basic path generated by BasicPath

Comments: ImprovePath scans the basic path in a recursive way from $\vec{b}$ to $\vec{a}$ using the *previous* field of the box class. At Step 3, it looks around the current box in order to find a possible shortcut. If the condition of Step 4 is satisfied, a shortcut is found and the path is modified. ♢

The shortcuts obtained for our test case for $\varepsilon = 0.4$ and for $\varepsilon = 0.2$ are represented on Figure 4 and 5 with thick segments.

6. Conclusion

In this paper, a new approach to deal with motion planning problems is presented. This approach uses interval analysis to characterize the configuration set (*i.e.* the set of all feasible configurations) using pavings. The box class has been extended to incorporate topological informations on the paving. This made it possible to develop a fast color-propagation algorithm that has been used to characterize a connected component of the configuration set. It was therefore easy to derive an algorithm to find a feasible path that links the start configuration point $\vec{a}$ to the final configuration point $\vec{b}$. Unfortunately, such a path is generally tortuous and an improvement avoiding unnecessary loops has been proposed. This improvement is not sufficient and more efficient strategies are under considerations.

The approach advocated here should easily be adapted to other problems related to topological analysis of sets, such as counting the number of connected component of a set. This could have some applications for example in the identifiability analysis of a model.

References

[1] L. Jaulin and E. Walter: "Set inversion via interval analysis", Automatica, 29(4), pp 1053-1064, 1993.

[2] L. Jaulin and E. Walter: "Guaranteed tuning, with application to robust control and motion planning", Automatica,32(8), pp 1217-1221, 1996.

[3] O. Khatib: "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", International Journal of Robotics Research, 5(1), pp 90-98, 1986.

[4] D.E. Koditschek: "Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations", Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh, NC, pp 1-6, 1987.

[5] T. Lozano-Pérez and M. Wesley: "An Algorithm for Planning Collision-free Paths among Polyhedral Obstacles". Communications of the ACM, Vol. 22, No. 10, pp 560-570, 1979.

[6] R.E. Moore: "Methods and Applications of Interval Analysis", SIAM, Philadelphia, 1979.

[7] N.J. Nilsson: "A Mobile Automaton: An Application of Artificial Intelligence Techniques", Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington D.C., pp 509-520, 1969.

[8] C. O'Dunlaing and C.K.Yap: "A Retraction Method for Planning the Motion of a Disc", Journal of Algorithms, 6, pp 104-111, 1982.