



# Un algorithme pour la planification de trajectoire basé sur le calcul ensembliste

Vincent Vigneron, Ryadh Kallel, Hichem Maaref

## ► To cite this version:

Vincent Vigneron, Ryadh Kallel, Hichem Maaref. Un algorithme pour la planification de trajectoire basé sur le calcul ensembliste. Rencontres Francophones sur la Logique Floue et ses Applications (LFA 2006), Sep 2006, Toulouse, France. pp.271-278, 2006. <hal-00203353>

**HAL Id: hal-00203353**

**<https://hal.archives-ouvertes.fr/hal-00203353>**

Submitted on 9 Jan 2008

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Un algorithme pour la planification de trajectoire basé sur le calcul ensembliste

## An interval calculus based algorithm for motion planning

Vincent Vigneron<sup>1,2</sup>, Ryadh Kallel<sup>2</sup>, Hichem Maaref<sup>1</sup>

<sup>1</sup>IBISC CNRS FRE 2873

40 rue du Pelvoux

91020 Evry Cedex, France

vvigne,maaref@cemif.univ-evry.fr

<sup>2</sup>MATISSE-SAMOS CNRS UMR 8595

90, rue de Tolbiac

75634 Paris cedex 13, France.

vigneron,kallel@univ-parisl.fr

### Résumé

Nous proposons dans cet article une solution originale et *garantie* à la “planification de trajectoire” en utilisant une approche ensembliste. Le système est capable de traiter des tâches compliquées dans un environnement encombré d’obstacles, avec pour seule entrée la géométrie de l’environnement, un point de départ et un point d’arrivée. La tâche de planification est résolue en combinant le calcul par intervalle et la théorie des graphes. La méthode est illustrée sur un cas idéalisé dans lequel le robot est choisi circulaire et les obstacles composés de figures géométriques simples.

**Mots-clés** Planification de trajectoire, intervalles, chemin flou,  $\alpha$ -coupe

### Abstract

*A new path planner is presented that automatically computes the collision-free trajectories for mobile robots in static workspaces. The implemented planner is capable of dealing with complicated tasks in an environment cluttered with obstacles and needs as input the geometry of the environment, the initial and the goal configuration. The planning is solved by combining interval and tools from graph theory. This will be illustrated by a planar test case where the robot is circular and the obstacles consist of line segments, circles and rectangles.*

**Keywords** Motion planning, interval, fuzzy path,  $\alpha$ -cut

## 1 Introduction

La recherche de trajectoire dans un environnement connu a été étudiée par de nombreux chercheurs [5, 6]. La plupart des approches actuelles de planifications de trajectoires sont basées sur le concept d’*espace de configuration* ou  $\mathcal{C}$ -espace

(cf. Lozano-Pérez *et al.* [10]). Le nombre de paramètres indépendants nécessaires pour désigner la configuration d’un objet correspond à la dimension de l’espace des configurations. L’espace des configurations *faisables*  $\mathcal{S} \subset \mathcal{C}$  contient seulement les vecteurs de configuration pour lesquels l’objet respecte les contraintes établies. Le problème de recherche de trajectoire formulée dans l’espace de configuration revient à trouver le chemin inclu dans  $\mathcal{S}$ , partant d’un point de départ  $A$  et rejoignant une position d’arrivée  $B$ . Plusieurs approches pour résoudre ce problème sont basées sur l’utilisation de fonctions potentielles, introduit par Khatib [9]), tandis que d’autres sont basées sur la subdivision de l’espace de configuration [4]. Cette approche découpe l’espace de configuration  $\mathcal{C}$  en zones d’appartenance, de non-appartenance ou incertaines. Les méthodes existantes utilisées pour décider si une boîte est incluse ou exclue de l’espace de configuration faisable sont limitées à une faible classe de problèmes [10].

L’*analyse par intervalles* (ApI) est capable de prouver qu’une boîte donnée est incluse ou exclue de  $\mathcal{S}$  pour une grande classe de problèmes, et, dans ce contexte particulier, en subdivisant l’espace de configuration avec des méthodes classiques basées sur la subdivision de l’espace de configuration. Il faut noter que l’ApI a déjà été utilisée pour la recherche de chemins paramétriques dans [8], mais ces méthodes requièrent un modèle paramétrique souvent délicat à estimer et sont donc limitées à des modèles de chemins à faible dimension.

Le papier est organisé comme suit. La section 2 rappelle les notions de base pour la construction d’un graphe associé au problème de recherche de trajectoire. En dehors du fait que le test utilisé pour décider de la faisabilité d’une boîte est basé sur l’ApI, le reste de la méthode est assez classique [1, 2]. La section 3 détaille un algorithme qui caractérise  $\mathcal{S}$  au moyen de sous-pavés avant de rechercher un chemin faisable. L’étape de construction du chemin est discutée dans la section 4, et plusieurs variantes sont proposées dont une incluant une *fuzzification* des trajec-

toires.

## 2 Construction du $\mathcal{S}$ -graphe

### 2.1 Notation et terminologie

En robotique mobile, l'environnement est une zone de travail 2D notée  $\mathcal{W}$  comportant  $q$  obstacles statiques  $\mathcal{B}_i (i = 1, \dots, q)$ . Soit  $\mathcal{CB}$  l'espace occupé par des obstacles défini par  $\mathcal{CB} \subset \mathcal{C}$  comme l'ensemble de toutes les configurations de système où deux corps ou plus de  $\{\mathcal{B}_1, \dots, \mathcal{B}_q, \mathcal{M}\}$  intersectent, *i.e.* les positions qui ne sont pas physiquement légales.

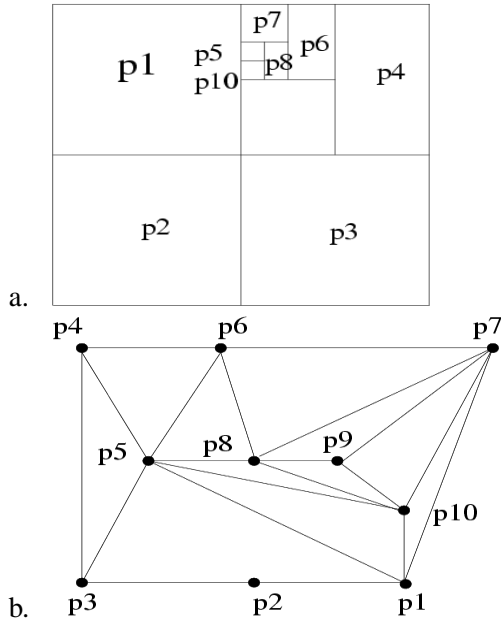


Figure 1 – (a) Pavage  $\mathcal{P}$  avec 10 boîtes. (b) Graphe  $\mathcal{G}$  associé au pavage  $\mathcal{P}$ .

Une caractérisation garantie de l'espace faisable de configuration, notée  $\mathcal{S}$ , peut être obtenue en utilisant une *discrétisation en graphes*, qui consiste à associer un graphe de l'espace d'état et sa représentation pour la planification de chemin.

Un pavage *régulier* d'une boîte  $[p_0]$  est un ensemble  $\mathcal{P}$  de boîtes sans recouvrement, par exemple  $\mathcal{P} = \{[p_1], \dots, [p_{10}]\}$ , dont l'union est égale à  $[p_0]$  telle que chacune de ces boîtes peut être obtenue à partir d'une succession finie de bisections (voir Fig. 1.a). Un tel *sous-pavage* peut être obtenu en utilisant un algorithme de "subdivision" tel que SIVIA [7] qui approxime n'importe quel ensemble compact de manière *garantie* sous forme de sous-pavés réguliers pouvant être facilement manipulés par un ordinateur sous forme.

Un graphe  $\mathcal{G} = \{V, E\}$  est constitué d'un ensemble non vide  $V$  de sommets et un ensemble

$E$  de paires de sommets non ordonnés de  $V$  appelées *arcs*. Si  $\nu_a$  et  $\nu_b$  sont deux sommets du graphe, l'arc associé à la paire  $(\nu_a, \nu_b)$  est noté  $ab$ . Les sommets forment l'ensemble des configurations du robot dans l'espace  $\mathcal{F}$  des configurations libres :  $\mathcal{F}$  est le sous-ensemble de  $\mathcal{C}$ , formés des positions libres du robot, c.-à-d.  $\mathcal{F} = \mathcal{C} \setminus \cup_O O^*$ , où, pour un obstacle  $O$ ,  $O^*$  est défini par les positions pour lesquelles le robot intersecte  $O$ .

Un chemin dans  $\mathcal{G}$  est une séquence de  $k$  sommets  $(\nu_1, \dots, \nu_k)$  tel que pour tout  $i, 1 \leq i \leq k$ , l'arc  $(\nu_i, \nu_{i+1})$  appartient à  $E$ . Le chemin est un cycle si  $\nu_k = \nu_1$ . Deux sommets  $\nu_i$  et  $\nu_j$  de  $\mathcal{G}$  sont voisins si l'arc  $(\nu_i, \nu_j)$  existe dans  $E$ . Un sous-graphe de  $\mathcal{G}$  est un graphe dont les sommets et les arcs appartiennent à  $\mathcal{G}$ .

Chaque pavage ou sous pavage  $P$  d'une boîte  $[p_0]$  peut être représenté par un graphe  $\mathcal{G}$ . Chaque sous boîte  $[p_i]$  de  $P$  est associé à un sommet  $\nu_i$  de  $\mathcal{G}$ . Si deux boîtes  $[p_i]$  et  $[p_j]$  sont adjacentes dans  $P$ , alors l'arc  $(\nu_i, \nu_j)$  existe dans  $\mathcal{G}$ . Par exemple, le graphe  $\mathcal{G}$  associé au pavage de la Fig. 1.a, est donné par la Fig. 1.b.

Tableau 1 – Algorithme du plus court chemin.

ALGORITHM ShortestPath	
Inputs :	$\mathcal{G}, s, g$ ; Output : $\mathcal{L}^0$ ;
1.	Pour tous les sommets $v \in V$ , faire $d(v) := \infty$
2.	$d(s) := 0; d_{\min} := 0;$
3.	Faire
4.	si $\mathcal{G}(d_{\min}) = \emptyset$ alors $\mathcal{L} := \emptyset$ , retourner('ECHEC') ;
5.	$d_{\min} := d_{\min} + 1$ ;
6.	Pour tous les sommets $v \in \mathcal{G}(d_{\min} - 1)$
7.	Pour tous les voisins $w$ de $v \in \mathcal{G}$
	avec $d(w) := \infty$ , faire $d(w) :=$
	$d_{\min}$
8.	Fin Pour
9.	Fin Pour
10.	jusqu'à $d(v_b) \neq \infty$
11.	$\ell := d(g); v_\ell := g$
12.	Pour $i := \ell - 1$ à 0, Faire
13.	choisir un voisin $v_i$ de $v_{i+1}$ tel que $d(v_i) := i$ ;
14.	Fin Pour
15.	$\mathcal{L}^0 := \{s, v_1, v_2, \dots, v_{\ell-1}, g\}$

### 2.2 Le chemin le plus court

Plusieurs algorithmes ont été proposés pour résoudre le problème de recherche du plus court chemin entre deux sommets  $\nu_a$  et  $\nu_b$  dans un graphe  $\mathcal{G}$ , le plus efficace est l'algorithme proposé par Dijkstra [3]. Bien qu'il ait été initialement développé pour des graphes orientés pon-

dérés (graphes avec des arcs orientés), une version simplifiée est présentée dans le tableau 1 (ShortestPath). Pour chaque sommet  $\nu$  de  $\mathcal{G}$  est associé un entier  $d(\nu)$  représentant le nombre minimum d'arcs entre les sommets  $\nu_a$  et  $\nu$ . On définit  $\mathcal{G}(i)$  avec  $i \in \mathbb{N}$ , l'ensemble des sommets de  $\mathcal{G}$  tel que  $d(\nu) = i$ . Si l'algorithme ShortestPath retourne ECHEC, alors  $\nu_a$  et  $\nu_b$  ne sont pas reliés dans le graphe  $\mathcal{G}$ . Sinon, l'algorithme retourne le chemin le plus court. En lançant ShortestPath ( $\mathcal{G}, \nu_1, \nu_6$ ), avec le graphe de la Fig. 1.b, on obtient par exemple  $d(p_1) = 0, d(p_2) = d(p_5) = d(p_{10}) = d(p_7) = 1, d(p_3) = d(p_4) = d(p_6) = d(p_8) = d(p_9) = 2$ . L'algorithme retourne le chemin  $\{p_1, p_5, p_4\}$ .

### 2.3 Solution ensembliste

Les étapes nécessaires à la résolution du problème de motion planning sont les suivantes :

1. Déterminer l'espace des configurations faisables  $\mathcal{S}$ .
2. Transformer l'espace l'ensemble des configuration de  $\mathcal{S}$  en un graphe.
3. Appliquer un algorithme de recherche du chemin le plus court.

L'apport de l'ApI se situera au niveau de l'étape 1. Pour cette étape, nous utiliserons un algorithme de partitionnement de l'espace de configuration (cf. tableau 2) reprenant le principe de l'algorithme SIVIA. L'algorithme de partitionnement prend comme paramètres d'entrées :

$[t](\cdot)$  : Une fonction réalisant un test d'inclusion,

$A$  : La configuration de départ,

$B$  : La configuration d'arrivée,

$[p_0]$  : Le pavé initial de recherche,

$\epsilon$  : La précision de l'algorithme.

On définit Graph( $P$ ) comme une fonction générant le graphe associé au pavage  $P$  et Sommet( $P$ ) une fonction convertissant la boîte  $[p]$  en un sommet du graphe  $\mathcal{G}$ .

L'algorithme présenté Tab. 2 construit deux sous pavages  $P^-$  et  $P^+$  qui satisfont  $P^- \subset \mathcal{S} \subset P^+$  et utilise une pile pour stocker toutes les boîtes qu'il reste à étudier. Les graphes  $\mathcal{G}^+$  et  $\mathcal{G}^-$  associés à  $P^-$  et  $P^+$  sont alors construits. L'algorithme choisit ensuite deux boîtes  $[p_s]$  et  $[p_g]$  de  $P^+$  tel que  $s \in [p_s]$  et  $g \in [p_g]$ . Il faut noter que deux boîtes ou plus peuvent être acceptables pour  $[p_s]$  et  $[p_g]$  si  $s$  et  $g$  sont situés à la frontière d'une boîte de  $P^+$ . Dans un tel cas de figure, l'algorithme choisit la première boîte qu'il trouve. On note  $\nu_s$  et  $\nu_g$  les deux sommets de  $\mathcal{G}^+$  associés à  $[p_s]$  et  $[p_g]$ . L'algorithme de partitionnement (Tab. 2) utilise l'algorithme

Tableau 2 – Algorithme de partitionnement de l'espace de configuration.

ALGORITHM Partitionnement	
<b>Inputs :</b> $[t](\cdot), s, g, [p_0], \epsilon$	
1.	si $[t](s) \neq \text{vrai}$ ou $[t](g) \neq \text{vrai}$ , retourner('ERREUR : $a$ et $b$ doivent être des configurations faisables')
2.	si $a \notin [p_0]$ ou $b \notin [p_0]$ retourner('ERREUR : $a$ et $b$ doivent appartenir à $[p_0]$ ')
3.	Pile = $\{[p_0]\}$ , $\Delta P = \emptyset$ , $P^- = \emptyset$ ;
4.	Tant que Pile = $\emptyset$
5.	$[p]$ =Dépiler Pile
6.	si $[t]([p]) = \text{vrai}$ , $P^- = P^- \cup [p]$
7.	si $[t]([p]) \neq \text{faux}$ ou $w([p]) \leq \epsilon$ , $\Delta P = \Delta P^- = P^- \cup [p]$
8.	si $[t]([p]) \neq \text{faux}$ et $w([p]) > \epsilon$ ,
9.	Bisection( $[p]$ ), Empiler les 2 boîtes résultantes
10.	Fin Tant que
11.	$P^+ = P^- \cup \Delta P$ , $\mathcal{G}^+ = \text{Graph}(P^+)$ , $\mathcal{G}^- = \text{Graph}(P^-)$ , $s = \text{Sommet}([p_s])$ où $[p_s] \in P^+$ et $s \in [p_s]$ $g = \text{Sommet}([p_g])$ où $[p_g] \in P^+$ et $g \in [p_g]$
12.	$\mathcal{L}^+ = \text{ShortestPath}(\mathcal{G}^+, \nu_s, \nu_g)$ .
13.	si $\mathcal{L}^+ = \emptyset$ , retourne('PAS DE CHEMIN')
14.	si $\nu_s \notin \mathcal{G}^-$ ou $\nu_g \notin \mathcal{G}^-$ , retourne('Erreur : $\nu_s$ et $\nu_g$ n'appartiennent pas à $\mathcal{G}^-$ ')
15.	$\mathcal{L}^- = \text{ShortestPath}(\mathcal{G}^-, \nu_s, \nu_g)$ .
16.	si $\mathcal{L}^- = \emptyset$ retourne $\mathcal{L}^-$
17.	sinon retourne('Erreur')

ShortestPath pour obtenir un chemin  $\mathcal{L}^+$  de  $\mathcal{G}^+$  allant de  $\nu_s$  à  $\nu_g$ . Si aucun chemin n'est trouvé, cela signifie qu'il n'existe aucun chemin dans  $P^+$  de  $\nu_s$  vers  $\nu_g$  et l'algorithme retourne "PAS DE CHEMIN". Si un chemin est trouvé, l'algorithme ShortestPath est de nouveau appelé pour trouver un chemin  $\mathcal{L}^-$  de  $\mathcal{G}^-$  qui relie  $\nu_s$  à  $\nu_g$ . La raison de réitérer l'algorithme est que  $\mathcal{L}^+$  était généré à partir d'un pavage contenant l'ensemble des boîtes vérifiant la relation  $[t]([p]) = \text{vrai}$ , mais aussi les boîtes *ambiguës*. Par conséquent, il est possible que le chemin  $\mathcal{L}^+$  passe par des sommets appartenant à des boîtes ambiguës qui peuvent potentiellement être invalides. Il faut alors relancer l'algorithme sur le sous pavage ne contenant que des boîtes vérifiant  $[t]([p]) = \text{vrai}$ , c'est-à-dire  $P^-$ . L'intérêt de calculer  $\mathcal{L}^+$  repose sur le fait que le sous pavage  $P^-$  dépend de la précision  $\epsilon$ . Ainsi, en fonction de celle-ci,  $P^-$  sera plus ou moins *large*, ce qui va jouer sur la résolution du problème. Par

contre, s'il existe un chemin allant de  $\nu_s$  à  $\nu_g$ , on est sûr de le trouver à partir de  $P^+$ . Par conséquent, si l'on trouve un chemin dans  $P^+$  mais pas dans  $P^-$ , il suffira d'augmenter la précision pour trouver le chemin exact.

Initialement, le graphe  $\mathcal{G} = \{V, E\}$  est vide. Puis, récursivement, un ensemble de configurations faisables est généré et ajouté à  $V$ . Pour chacun de ces noeuds  $v$ , on choisit plusieurs noeuds de  $V$  et on essaye de connecter  $v$  à chacun d'eux en utilisant la fonction SHORTESTPATH. La configuration obtenue est ensuite vérifiée : si elle est sans collision, elle est ajoutée à  $E$ , sinon, elle est ignorée. Chaque fois que ce planificateur réussit à calculer un chemin faisable entre  $v$  et un sommet  $p_i$ , l'arc  $(v, p_i)$  est ajouté à  $E$ . Dans la mise en oeuvre générale (considéré section 3.2), les tests de collision utilisent des règles de géométrie simples.

L'algorithme est ainsi capable de connecter n'importe quelle configuration de départ et d'arrivée très rapidement, pourvu que le découpage du plan soit assez dense, mais pas nécessairement régulier : il peut être localement dense si plus de noeuds sont nécessaires dans le voisinage d'obstacles par exemple. De plus les chemins locaux n'ont pas besoin d'être mémorisés car leur "recalcul" est peu coûteux.

### 3 Application à la navigation

#### 3.1 Paramétrisation

Le problème sera le suivant : il s'agit de déplacer un robot entre deux positions sans qu'il n'entre en collision avec le décor. La plupart des solutions existantes sont complexes et ne peuvent pas être généralisées. Le  $\mathcal{C}$ -espace est défini par rapport aux paramètres libres  $(x_i, y_i, \theta)$  du robot, représenté par un cercle de rayon  $\rho$ . La difficulté est accrue par l'imprécision de la carte. La figure 2 présente les paramètres du robot.

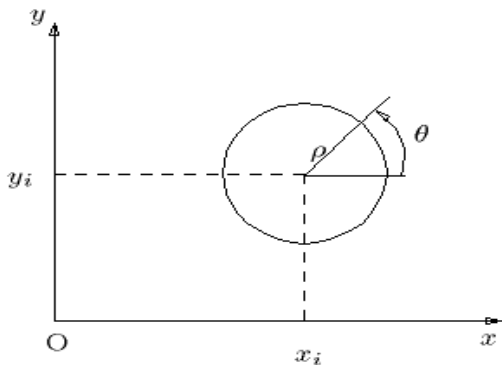


Figure 2 – Paramètres du robot mobile  $(x_i, y_i, \theta)$ .

L'idée serait alors de définir l'espace de configuration faisable par rapport aux paramètres non figés du robot c.-à-d.  $(x_i, y_i, \theta)$ . Le problème est que  $\theta$  n'est pas facile à intégrer dans le partitionnement de l'espace. En effet : le robot est non-holonome, donc inclure ce paramètre dans le partitionnement de l'espace de configuration risquerait de rendre adjacent deux configurations qui ne respecteraient pas cette contrainte lors de la construction du graphe associé. La solution serait de tenir compte de cette contrainte lors de la création du graphe, mais on perdrait alors en généralité. Nous avons donc opté pour ne partitionner que dans le plan  $(x, y)$ .  $\theta$  sera obtenu après avoir calculé le chemin, le robot sera ainsi toujours orienté dans le sens de la trajectoire.

Nous avons choisi de permettre à l'utilisateur de saisir lui-même les obstacles à travers l'interface de l'application. 3 formes sont actuellement disponibles en position et taille quelconque : rectangles, ovales, et segments de droites.

Il reste maintenant à définir la *fonction d'inclusion*.

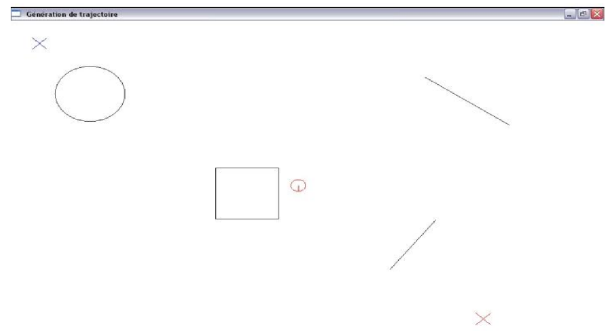


Figure 3 – Capture d'écran de l'application finale.

#### 3.2 Fonction d'inclusion

Le partitionnement de l'espace est basé sur la détermination des zones de l'espace où le mobile n'entre pas en collision avec les obstacles. Il faut donc définir une fonction d'inclusion intégrant ces critères, en réalisant un test d'inclusion pour chaque type d'obstacle. Nous commençons par un rappel élémentaire sur les tests d'inclusions. Soit  $\mathbb{A} \subseteq \mathbb{R}^2$ . Un test inclusion  $[t_A]$  pour  $\mathbb{A}$  satisfait

$$[t_A]([x]) = \begin{cases} 1 & \Rightarrow (\forall x \in [x], t_A[x] = 1) \\ 0 & \Rightarrow (\forall x \in [x], t_A[x] = 0) \end{cases} \quad (1)$$

Par exemple, considérons le test

$$t : \mathbb{R}^2 \rightarrow \{0, 1\} \\ t : (x_1, x_2)^T \rightarrow (x_1 x_2 + x_1^2 + x_2^2 \leq 5), \quad (2)$$

c.-à-d.

$$t(x) = \begin{cases} 1 & \text{si } x_1x_2 + x_1^2 + x_2^2 \leq 5, \\ 0 & \text{si } x_1x_2 + x_1^2 + x_2^2 > 5. \end{cases} \quad (3)$$

Le test d'inclusion minimal  $[t]$  associé à  $t$  est donné par :

$$[t_A]([x]) = \begin{cases} 1 & \text{if } \overline{x_1x_2} + \overline{x_1^2} + \overline{x_2^2} \leq 5, \\ 0 & \text{if } \overline{x_1x_2} + \overline{x_1^2} + \overline{x_2^2} > 5, \\ [0, 1] & \text{sinon.} \end{cases} \quad (4)$$

qui peut aussi s'écrire sous une forme plus concise  $[t]([x]) \Leftrightarrow ([x_1][x_2] + [x_1]^2 + [x_2]^2)$ .

**A. Test de collision entre le robot mobile et un rectangle.** On définit les sommets  $A$  et  $B$  comme étant les sommets opposés du rectangle considéré lors du test. Le robot mobile est représenté par sa boîte englobante comme le montre la figure 3, ainsi il y aura collision lorsque la boîte englobante et le rectangle se recouvrent.

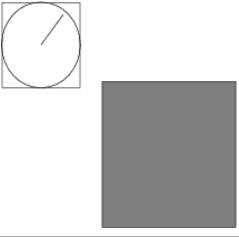
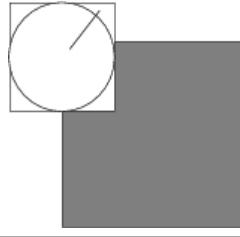
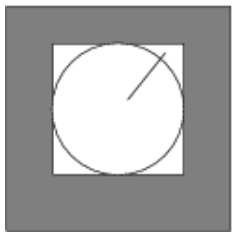
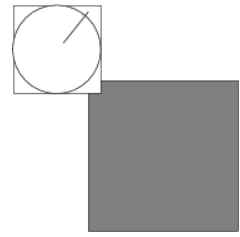
case 1 : no collision	case 2 : collision
	
case 3 : collision	case 4 : collision
	

Tableau 3 – Différent cas de figure lors du test de collision robot rectangle.

Dans la figure 3, le cas 1 est le seul où il n'y a pas de collision. L'inconvénient d'approximer le robot mobile à sa boîte englobante est que l'on considérera qu'il y a collision dans le cas de la Fig. 3.4 alors qu'en réalité, il n'y en a pas. L'avantage est que cette représentation correspond exactement à la notion de pavé. En effet, en ayant limité les paramètres à  $(x_i, y_i)$ , il s'agit de tester directement chaque pavé avec le rectangle sans aucune autre manipulation particulière. En notation ensembliste, il y a collision si  $[x_R] \cap [x_B - x_A] = \emptyset$  et  $[y_R] \cap [y_B - y_A] = \emptyset$ .

**B. Test de collision entre le robot mobile et un cercle.** C'est le cas le plus simple, en effet cela revient à un test cercle/cercle dont voici le principe.

Deux cercles  $C_1(O_1, \rho_1)$  et  $C_2(O_2, \rho_2)$  sont en collision lorsque  $d(O_1, O_2) \leq \rho_1 + \rho_2$ , où  $O_1$  et  $O_2$  sont les centres des cercles définis par leurs coordonnées resp.  $(x_{01}, y_{01})$  et  $(x_{02}, y_{02})$ ,  $\rho_1$  et  $\rho_2$  les rayons de ces cercles. Si  $d(\cdot)$  est la distance euclidienne, alors  $d(O_1, O_2) = \sqrt{(x_{01} - x_{02})^2 + (y_{01} - y_{02})^2}$ . D'un point de vue calcul ensembliste, l'intersection se traduit par  $d(O_1, O_2) \leq \rho_1 + \rho_2$ .

Afin d'optimiser le test, nous allons travailler sans racine carrée, car c'est une opération lente. Le test de collision devient : si  $([x_R] - [x_{02}])^2 - ([y_R] - [y_{02}])^2 \cap [0, (\rho_1 + \rho_2)^2] \neq \emptyset$ , alors l'événement collision est VRAI.

**C. Test de collision entre le robot mobile et un segment de droite.**

L'idée retenue est d'approximer le robot mobile à son centre et de travailler sur la notion de distance entre ce centre et le segment.

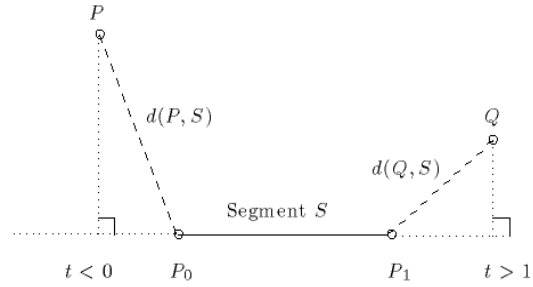


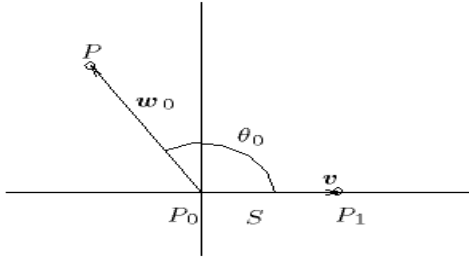
Figure 4 – Distance d'un point à une ligne.

L'avantage de cette approche est que la structure du robot mobile est en cercle. Par conséquent, il n'y aura collision que lorsque la distance sera inférieure au rayon du robot.

L'équation paramétrique d'une ligne est donnée par  $P(t) = P_0 + t(P_1 - P_0)$  où  $P_0, P_1$  sont 2 points de la ligne,  $\overrightarrow{P_0P_1}$  est la direction de la ligne et  $t \in \mathbb{R}$ . Dans le cas d'un segment de droite  $S$  dont les extrémités sont  $(P_0, P_1)$ ,  $0 \leq t \leq 1$  (Fig. 4). Le point le plus proche entre le point  $P$  et le segment de droite  $S$  est le projeté de  $P$  sur  $S$ , noté  $P(b)$ . Mais le projeté peut se trouver à l'extérieur du segment (cf. Fig. 4), et dans ce cas, la plus petite distance du segment à  $P$  est une extrémité du segment. Une façon simple de procéder consiste à examiner les angles entre  $\overrightarrow{P_0P_1}$  et les vecteurs  $\overrightarrow{P_0P}$  et  $\overrightarrow{P_1P}$  (Tab. 4).

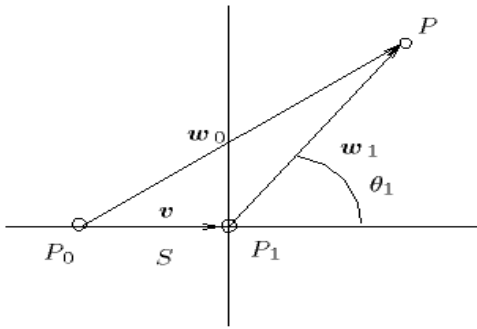
Si l'un ou l'autre de ces angles vaut  $90^\circ$ , alors l'extrémité du segment est  $P(b)$ , sinon il est d'un côté ou de l'autre de cette extrémité selon que cet angle est aigu ou obtus. Ces conditions sont facilement évaluées en testant le signe du produit

$$w_0 = P - P_0 \text{ and } \theta_0 \in [-180^\circ, 180^\circ]$$



$$\begin{aligned} w_0 \cdot v &\leq 0 \\ \Leftrightarrow |\theta_0| &\geq 90^\circ \text{ is obtuse} \\ \Leftrightarrow d(P, S) &= d(P, P_0) \end{aligned}$$

$$w_1 = P - P_1 \text{ and } \theta_1 \in [-180^\circ, 180^\circ]$$



$$\begin{aligned} w_1 \cdot v &\leq 0 \Leftrightarrow w_0 \cdot v \geq v \cdot v \\ \Leftrightarrow |\theta_1| &\leq 90^\circ \text{ is acute} \\ \Leftrightarrow d(P, S) &= d(P, P_1) \end{aligned}$$

Tableau 4 – Mathematical sketch to compute point-to-segment distance.

scalaire  $w_0 \cdot v$  ou  $v \cdot v$ , en posant  $v = (P_1 - P_0)$  et  $w = (P - P_0)$ .  $w_0 \cdot v$  et  $v \cdot v$  entrent dans le calcul de  $b$  dans l'Eq. 5 :

$$b = \frac{d(P_0, P(b))}{d(P_0, P_1)} = \frac{|w| \cos \theta}{|v|} = \frac{w \cdot v}{|v|^2} \quad (5)$$

La distance de  $P$  à la ligne est donnée par :

$$d(P, S) = |P - P(b)| = |w - bv| = |w - (w \cdot u)u| \quad (6)$$

où  $u$  est le vecteur directeur unitaire de  $S$ .  
Le pseudocode de l'algorithme est le suivant :

```

distance(Point P, Segment P0:P1)
{
  v = P1 - P0
  w = P - P0
  if ( (c1 = w.v) <= 0 )
    return d(P, P0)
  if ( (c2 = v.v) <= c1 )
    return d(P, P1)
  b = c1 / c2
  Pb = P0 + bv
  return d(P, Pb)
}

```

Ce calcul a 2 avantages : il est efficace pour *n'importe quelle dimension*, et il permet de calculer  $b$ .

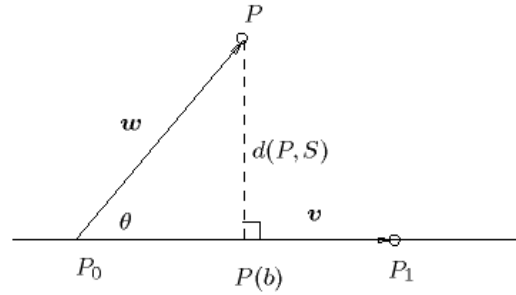


Figure 5 – Distance point-segment ( $0 \leq t \leq 1$ ).

## 4 Quelques résultats

La figure 6 présente l'interface de l'application et visualise un carré unitaire avec des obstacles et un robot  $R$  positionné *initialement* sur la croix rouge ( $\times$ ). On y retrouve tous les éléments cités dans les sections précédentes. Les expériences sont conduites dans un environnement simulé. La topologie est simple et de nombreux chemins permettent de rejoindre la croix bleue ( $\times$ ).

L'algorithme détermine toutes les routes possibles du robot et choisi la meilleure selon un certain *critère*, qui peut être celui de la distance parcourue minimale par exemple, avec une contrainte de sécurité maximale. Puis il calcule la liste des points *passants* du chemin qui approxime la trajectoire du robot.

La figure 6 présente le partitionnement de l'espace des configurations suivant notre algorithme pour un environnement donné. Les couleurs utilisées figure 6 correspondent aux propriétés suivantes :

- **rouge** : pavés vérifiant la contrainte d'*absence de collision*,
- **vert** : pavés *incertains*,
- **bleu** : pavés pour lesquels le robot entre *en collision*.

L'espace des configurations faisables est donc l'ensemble des pavés rouges.

La figure 7 affiche l'espace de configuration dans le cas d'un obstacle circulaire. Les déplacements du robot sont représentées par des lignes entre les centres des boîtes.

Il est intéressant de noter que la trajectoire trouvée, n'est pas la plus rapide. En effet, une meilleure trajectoire aurait consisté à "longer" l'obstacle, ceci est dû à la simplicité de l'algorithme reproduit Tab. 1 : dans cet algorithme, on considère que tous les arcs ont la même distance, ce qui conduit le programme à générer le chemin le plus court en terme de nombre de transitions. Ce qui explique que ce chemin ne s'approche pas des obstacles car, en leur voisinage, le pavage est

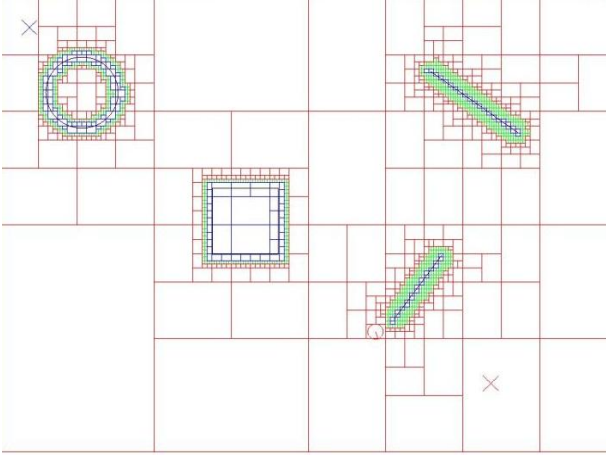


Figure 6 – Espace de configuration du problème du robot mobile.

plus dense, par conséquent contient plus de transition. Cette approximation était juste car on ne peut pas vraiment définir une notion de distance entre deux configurations, cependant nous avons décidé d'en définir une ce qui a conduit à la réécriture de l'algorithme du plus court chemin.

L'utilisation de cet algorithme a permis d'obtenir le résultat de la figure 8.

Comme prévu, le chemin le plus court consiste à longer l'obstacle. Dans les simulations des Fig. 9 et 10, des environnements plus complexes sont proposés. L'algorithme ShortestPath2 a permis d'obtenir l'espace des configurations faisables les plus prometteuses, et stoppe dès que le nombre requis de chemins faisables  $p$  a été trouvé. Sur la figure 9,  $p = 2$ .

**Fuzzification de trajectoire.** Dans [9], les obstacles à éviter sont représentés par un champ po-

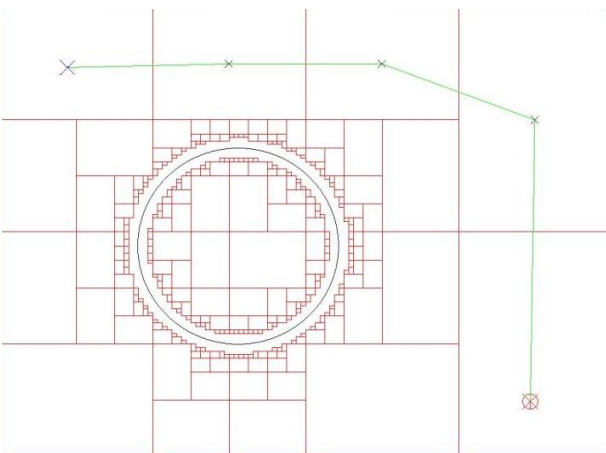


Figure 7 – Plus court chemin (nombre minimum d'arc).

Tableau 5 – Algorithme amélioré du plus court chemin.

ALGORITHM ShortestPath2	
Inputs :	$\mathcal{G}, s, g$ ;
Output :	Pile2 ;
1.	Pour tous les sommets $v \in \mathcal{G}$ , faire $d(v) := \infty$ et $Pred(v) = 0$
2.	$d(s) := 0$ , Pile1 = $\emptyset$
3.	Empiler(Pile1, $v_s$ ) ; fin=FAUX ;
4.	Tant que Pile1 $\neq \emptyset$ et fin==FAUX
5.	$v$ =élément dont la distance est la plus petite dans 'Pile1'
6.	Retirer(Pile1, $v$ )
7.	si ( $v = v_g$ ) alors fin=VRAI ;
8.	FinSi
9.	Pour tous les voisins $w$ de $v$ dans $\mathcal{G}$ avec $d(w) > d(v) + d(v, w)$
	Faire
10.	$d(w) = d(v) + d(v, w)$
11.	$Pred(w) = v$
12.	Empiler(Pile1, $w$ ) ;
13.	FinPour
14.	FinTantQue
15.	si $d(v_g) = \infty$ alors retourner('PAS DE SOLUTION') ;
16.	FinSi
17.	Pile2 = $\emptyset$ , $v = v_g$ ;
18.	Tant que $Pred(v) \neq 0$
19.	Empiler(Pile2, $v$ ) ;
19.	$v = Pred(v_g)$ ;
13.	FinTantQue

tentiel répulsif, et le but est représenté par un champ potentiel attractif. Selon la force générée par la somme de ces champs potentiels, l'objet est supposé atteindre la destination finale sans entrer en collision avec aucun obstacle.

Nous avons modélisé ce champ de potentiel par un *ensemble flou*. Pour une manipulation pratique de nos algorithmes de planification, des  $\alpha$ -coupes ont été associées à cet ensemble flou. Plus précisément, pour toute valeur  $\alpha \in [0, 1]$ , on définit la  $\alpha$ -coupe  $A_\alpha$  d'un sous-ensemble flou  $A$  de  $X$  comme le sous-ensemble  $A_\alpha = \{x \in X | f_A(x) \geq \alpha\}$ , de fonction caractéristique  $\chi_{A_\alpha}$  telle que  $\chi_{A_\alpha} = 1$  si et seulement si  $f_A(x) \geq \alpha$ . Dirigé par la force générée par ces potentiels, les trajectoires du robot  $\mathcal{R}$  sont différentes selon le niveau  $\alpha$  du *potentiel flou* (see Fig. 9) mais finissent par rejoindre la configuration désirée sans rencontrer d'obstacle.

## 5 Conclusion

L'algorithme est générique et pourra donc être appliqué à d'autres applications identiques : il suffit d'approximer l'objet à son *cercle englobant*. L'algorithme génère des trajectoires très rapidement et cela même avec un nombre impor-



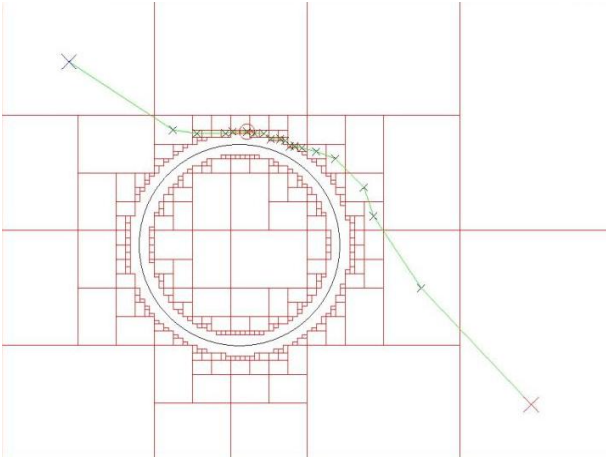


Figure 8 – Algorithme du plus court chemin revisité.

tant d'obstacles (sur des machines standard type PENTIUM IV, 1 GHz, 512 Mo RAM). Par contre, il peut s'avérer plus lent s'il y a trop d'obstacles. De plus, il ne reflète pas la "réalité", car il ne permet pas d'éviter les collisions entre des objets en mouvement, l'application ne fonctionnant que pour un environnement statique.

## Références

- [1] J.D. Boissonnat, B. Faverjon, and J.P. Merle. *Techniques de la robotique. Perception et planification*, volume 2. Hermes, 1988.
- [2] R.A. Brooks and T.A. Lozano-Pérez. A subdivision algorithm in configuration space for find path with rotation. *IEEE Trans. on Sys. Man and Cyber.*, 15(2) :224–233, 1985.
- [3] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math.*, 1 :269–271, 1959.
- [4] B. Faverjon and P. Tournassoud. A local based approach for path planning of ma-

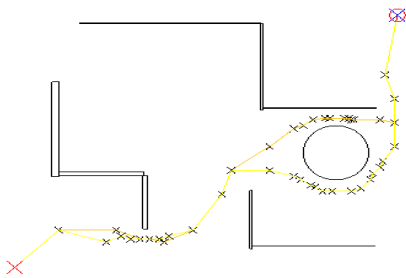
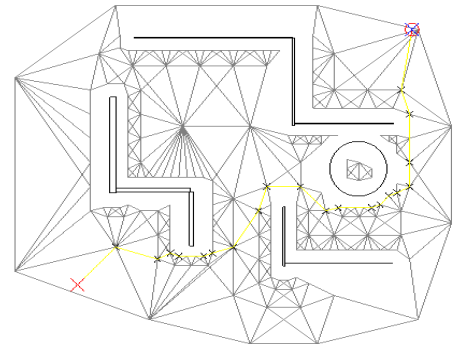
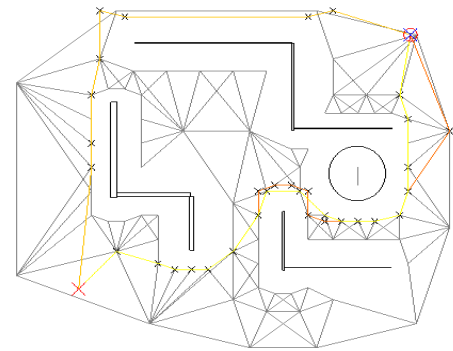


Figure 9 – Chemins multiples.



$\epsilon = 10$



$\epsilon = 20$

Figure 10 – Graphes obtenus pour des valeurs différentes du potentiel "repulsive" des obstacles.

nipulators with a high number of degrees of freedom. In *International Conference on Robotics and Automation*, pages 1152–1159, 1987.

- [5] K. Gupta and A. P. del Pobil. *Practical Motion Planning in Robotics : Current Approaches and Future Directions*. John Wiley and Sons, West Sussex, 1998.
- [6] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3) :219–291, september 1992.
- [7] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis*. Springer, Paris, 2001.
- [8] L. Jaulin and E. Walter. Guaranteed tuning, with application robust control and motion planning. In *Automatica*, volume 32, pages 1217–1221, 1996.
- [9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1) :90–98, 1986.
- [10] T. Lozano-Pérez. Spatial planning : a configuration space approach. *IEEE Trans. on Computers*, 32(2), 1983.