République Algérienne Démocratique et Populaire
Ministére de l'Enseignement Supérieur et de la Recherche Scientifique
Université A/Mira de Béjaia
Faculté de la technologie
Département De Génie Electrique

Mémoire de Master

En vue de l'obtention du diplôme de master en électronique

Option : Automatique

Thème

Planification de chemin par carte de route probabiliste : Cas d'une scène à deux dimensions et robots ponctuel ou polygonal

---

Presenté par :

BELAID Amine
HAMA Radouane

Composition du jury

M^{r}.K. MOKRANI	President
M^{r}.H. HADDAR	Promoteur
M^{r}.H. LEHOUCHE	Examinateur

Année Universitaire : 2015 / 2016

Remerciement

Nos remerciements vont tout premièrement ***à Dieu*** *tout puissant pour la volonté, la santé et la patience, qu'il nous a donnée durant toutes ces longues années.*

Nous exprimons nos profondes gratitudes à ***nos parents*** *pour leurs encouragements, leur soutien et pour les sacrifices qu'ils ont enduré.*

Nous tenons également à exprimer nos vifs remerciements à notre encadreur Monsieur " ***Haddar Hocine*** *" pour avoir d'abord proposé ce thème.*

Nous remercions ***les membres de jury*** *d'examen pour l'honneur qu'ils nous font en participant au jugement de ce travail.*

Nous tenons à remercier vivement tous ceux qui nous ont aidés de près ou de loin à accomplir ce travail.

Nous remercions vont aussi à tous ***les enseignants*** *du département d'électronique qui ont contribué à notre formation.*

Enfin nous tenons à exprimer notre reconnaissance à tous ***nos amis*** *et collègues pour leur soutien moral et matériel...*

Dédicaces

Je dédie ce travail :

A ma chère Mère.

A mon cher père.

A mes sœurs, et à mes frères : **Abdelwahab** et **Said**.

A mon cher Ami : **Slimane**.

A mon cher Ami et Co-binôme **Radouane** pour tous les moments de joies et de peines qu'on a passés ensemble.

A mes meilleurs amis tawwat de bejaia, les étudiants et les commerçants.

A mes amis de la cité universitaire 1000 lits surtout : **Abdelghani**, **Najib**, **Faical**, et de la cité targa-ouzomour surtout : **Brahim**, **Halim**, **Yacine**.

A tous qui m'ont connu et aidé de prés et de loin dans la réalisation de ce travail.

A tous les proches que j'ai mentionnés et les autres que j'ai oubliés veuillez m'excuser.

Amine

Dédicaces

A ceux qui m'ont donné la vie, symbole de beauté, de fierté, de sagesse et de patience.

A ceux qui sont la source de mon inspiration et de mon courage, à qui je dois de l'amour et la reconnaissance.

A ma très chère Mère et à mon cher Père **Abdelkader**.

A mes frères **Fouâd**, **Rahim** et ma sœurs **Lydia**.

A ma chère fiancer **Rosa** et sa famille surtout :**Mouhamad**, **Ayoub**, **Ryma**,**Amina**.

A mon cher Ami et Co-binôme **Amine** pour tous les moments de joies et de peines qu'on a passés ensemble, A sa Famille aussi.

A tous mes amis surtout : **Brahim**,**Halim**, **Abdellah**, **Ali**, **Youcef**, **Lhadi**,**Nadjib** ,**Messaoud**...

A tous mes collègues de promotion.

A tous les travailleurs de la cité universitaire targa-ouzomour.

A tous les proches que j'ai mentionnés et les autres que j'ai oubliés veuillez m'excuser.

Je vous remercie tous

Radouane

Table de matières

Table des figures

Liste des tableaux

Introduction générale

Le mythe fascinant de la machine crée par l'homme qui effectuerait toutes les tâches a nourri pendant longtemps l'imaginaire du grand public. Ce mythe symbolise la volonté de l'homme de faire exécuter des tâches fastidieuses par des machines appelées robots. L'évolution de ces machines est généralement classifiée suivant des générations reflétant les potentialités offertes par ces dernières. Les robots de première génération exécutent uniquement une suite de mouvement pré-enregistrés (chariot filoguidé). Ceux de la deuxième génération sont dotés de fonction élémentaire de perception leur permettant de se diriger de manière simple dans leur environnement. Les développements récents de la robotique visent à accroître l'autonomie de ces systèmes. Cette notion d'autonomie représente une composante importante caractérisant les robots de la troisième génération.

Un système robotique agit par le mouvement dans un monde physique. Sa capacité de *planification de mouvement* apparaît ainsi comme une composante essentielle de l'autonomie d'un robot.

Dans sa version la plus simple, la planification de mouvement s'intéresse au calcul automatique de chemins sans collision pour un système mécanique (robot mobil, bras manipulateur, personnage animé,...) évoluant dans un environnement encombré d'obstacles. La solution de ce problème, appelée *trajectoire* se présente comme une suite continue de situations géométriques successivement occupées par le robot durant son déplacement.

Les premières solutions au problème de planification de mouvement remontent au début des années 1980 avec l'introduction de la notion *d'espace des configurations* [1] : Une configuration regroupe l'ensemble des paramètres permettant de localiser un système mécanique dans son environnement. La planification de mouvement pour un système est alors ramenée au problème de la planification de mouvement d'un point dans cet espace des configurations. De nombreuses approches ont par la suite été proposées pour la résolution

de diverses instances de ce problème, visant à construire ou explorer efficacement cet espace. Face à la complexité combinatoire du problème, les approches probabilistes [2][3] apparues depuis une dizaine d'années permettent aujourd'hui de résoudre de nombreux problèmes qui restent hors de portée des méthodes *déterministes* initialement développées, en particulier lorsqu'il s'agit d'explorer des espaces de recherche hautement dimensionnés. Leur champ d'application dépasse aujourd'hui la seule robotique : elles intéressent des domaines aussi diversifiés que la CAO (Conception Assistée par Ordinateur), l'animation graphique ou la biochimie [4].

L'objectif de ce mémoire est de réaliser sous MATLAB un planificateur de chemin basé sur la méthode de carte de route probabiliste pour un corps polygonal, effectuant des translations et des rotations, dans une scène en 2D où les obstacles sont des polygones non convexes, et ceci en implémentant trois méthodes d'échantillonnages (deux traitent le problème des passages étroits), et aussi avec deux planificateurs locaux, et en utilisant deux méthodes de recherche du plus court chemin dans un graphe.

Ainsi, et en vue de présenter de manière claire notre travail, le présent mémoire est organisé de la manière suivante :

Dans le premier chapitre, nous présenterons une introduction sur la modélisation des robots et les méthodes de planification de trajectoire.

Le deuxième chapitre présente deux outils très importants pour les planificateurs de mouvement probabilistes, à savoir la théorie des graphes et le test de collision.

Dans le troisième chapitre nous allons présenter les détails des méthodes basées sur la carte de route probabiliste, avec les trois méthodes d'échantillonnages (uniforme, gaussien, par pont) et les deux planificateurs locaux que nous avons utilisés dans notre réalisation.

Dans le quatrième chapitre, nous allons présenter notre réalisation d'un planificateur de chemin basé sur la carte de route probabiliste dans lequelle nous avons utilisé trois méthodes d'échantillonnages, deux planificateurs locaux et deux méthodes de recherche du plus court chemin, éléments qui ont fait l'objet d'explication dans les chapitres précédents.

La conclusion vient présenter une brève synthèse du travail réalisé en récapitulant les principaux fondements théoriques derrière la méthode et en proposant des perspectives qui seraient susceptibles d'aboutir à de nouvelles améliorations.

Chapitre 1

Modélisation des robots et planification de mouvement

1.1 Introduction

Le mot « *robot* » a été dérivé du mot tchèque « *robota* », ce qui signifie le travail forcé, ce mot « *robot* » est devenu populaire en 1921 à cause d'un jeu nommé "*Universal Robots de Rossum*" qui a été créé par l'écrivain tchécoslovaque *Karel Capek*. Dans le jeu, un scientifique appelé *Rossum* a créé des machines comme des humaines et ces machines se révoltèrent, et tuèrent leurs maîtres humains, et ont pris le contrôle du monde [5].

La société américaine, Unimate, créé en 1962 par *Joseph Engelberger* et *George Devol*, a été la première entreprise à produire effectivement des robots industriels.

D'un point de vue technique, un robot est un système alimenté en énergie qui évolue dans un environnement statique ou dynamique, il est formé d'un microcontrôleur ainsi que d'un ou plusieurs capteurs et actionneurs. Et d'un point de vue mécanique, un robot est un ensemble des corps qui sont souvent rigides.

Pour que le robot réalise des tâches, il est obligé de bouger, alors le mouvement est la problématique commune à tous les robots. Le mouvement est la forme visible d'actions pour l'atteinte des résultats dans le monde réel. D'un point de vue physique, un mouvement est le résultat de la force ou du couple appliqué à un corps. Si ce corps est fixé à un point particulier, une force appliquée fera tourner le corps autour d'un axe

qui passe par ce point. Par conséquent, il est important de comprendre la description mathématique du mouvement en termes de déplacement, de vitesse et d'accélération. Cela constitue une base solide qui aidera également à étudier certains sujets importants dans la robotique, comme *la planification de mouvement et de contrôle.*

1.2 Représentation d'un corps rigide dans l'espace

La position d'un point dans l'espace par rapport à un repère peut exprimer par ces coordonnés selon x et selon y et selon z :

$$M = \begin{vmatrix} x \\ y \\ z \end{vmatrix}$$

Mais pour un cops rigide on a besoin de la position et d'orientation par rapport à un repère de référence (fixe), pour cela on associe un repère mobile à ce corps, et on étudie la position et l'orientation du repère mobile par rapport au repère fixe.

Soit les deux repères $o_0x_0y_0z_0$ et $o_1x_1y_1z_1$, tel que le repère 0 est fixe et le repère 1 mobile, Lorsque le repère 1 est tourné de l'angle θ autour de l'axe z_0 (Figure 1.1), on construit la matrice de rotation R_1^0 comme suit :

$$R_1^0 = \begin{bmatrix} x_1.x_0 & y_1.x_0 & z_1.x_0 \\ x_1.y_0 & y_1.y_0 & z_1.y_0 \\ x_1.z_0 & y_1.z_0 & z_1.z_0 \end{bmatrix} \tag{1.1}$$

Ainsi, les colonnes de la matrice de rotation sont obtenues par la projection d'un axe du repère mobile 1 sur les axes du repère fixe 0. Et on dites R_1^0 est la matrice de rotation du repère mobile par rapport au repère fixe. Pour le cas de la figure 1.1 on a :

$$R_1^0 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = R_{z_0,\theta} \tag{1.2}$$

Les matrices de rotation en 3D appartiennent au *groupe orthogonal spécial*,... $R \in SO(3)$, qui ont des propriétés spéciales, comme $R^{-1} = R^T, \det(R) = 1$

Cette matrice de rotation ne sert pas seulement à exprimer l'orientation mais peut aussi être utilisée pour transformer les coordonnées d'un point dans un repère vers un autre. En connaissant les coordonnées p^1 d'un point dans le repère mobile, ces coordonnés p^0 dans le repère fixe sont :

$$p^0 = R_1^0 \, p^1 \tag{1.3}$$

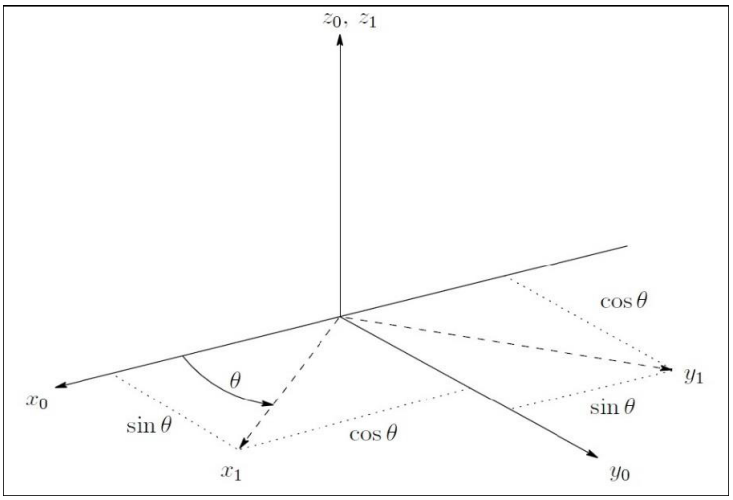

FIGURE 1.1 – Le repère 0 fixe et le repère 1 mobile, tourné de l'angle θ autour de l'axe z_0.

Dans le cas où plusieurs rotations sont faites, la matrice de rotation totale est un produit de tous ces matrices, et ceci en respectant deux règles :

- Si la rotation est faite autour d'un axe fixe on multiple à gauche (pré-multiplication).

  Exemple : Soient 3 rotations successives :

  $$R_{x,\alpha} \to R_{z,\theta} \to R_{y,\phi} \tag{1.4}$$

  La matrice de rotation résultante correspond à une pré-multiplication des 3 matrice :

  $$R = R_{y,\phi} \cdot R_{z,\theta} \cdot R_{x,\alpha} \tag{1.5}$$

- Si la rotation est faite autour d'un axe mobile on multiple à droite (post-multiplication).

Exemple : Soient 3 rotations successives :

$$R_{y,\phi} \rightarrow R_{z',\theta} \rightarrow R_{x',\alpha} \tag{1.6}$$

La matrice de rotation résultante correspond à une post-multiplication des 3 matrice :

$$R = R_{y,\phi} \cdot R_{z',\theta} \cdot R_{x',\alpha} \tag{1.7}$$

Une matrice de rotation peut être paramétré en utilisant trois angles. La convention commune est d'utiliser les angles d'Euler (φ, θ, ψ), qui correspondent à des rotations successives autour des axes x, y et z. La matrice de rotation résultante est donné par

$$R(\varphi, \theta, \psi) = R_{z,\psi} \cdot R_{y,\theta} \cdot R_{x,\varphi} \tag{1.8}$$

Roulis, tangage et lacet sont des angles semblables, sauf que les rotations successives sont effectuées par rapport au repère fixe, au lieu d'être effectuées par rapport au repère courant.

Si le corps rigide fait une rotation et une translation en même temps, un vecteur de translation P est concédé avec la matrice de rotation R qui forme une matrice de transformation suivant :

$$T = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}; R \in SO(3); P \in \mathbb{R}^3 \tag{1.9}$$

La matrice de transformation T est appelé matrice de transformation *homogène*, qui appartient au $SE(3)$ tel que

$$SE(3) = \mathbb{R}^3 \times SO(3) \tag{1.10}$$

1.3 Modèle géométrique direct et inverse

Un robot manipulateur est constitué d'un ensemble d'organes reliés entre eux par des articulations. Les articulations peuvent être rotoïdes ou bien prismatique, comme elles peuvent être plus complexes. L'objectif du modèle géométrique direct est de déterminer la position et l'orientation de l'effecteur à partir des variables des articulations. Et vis-versa pour le modèle géométrique inverse [6].

Pour un robot mobile dans le plan ou bien dans l'espace son modèle géométrique et simple car il n'y a que deux repères, un repère fixe et un repère mobile associé au robot. Mais pour les robots manipulateurs le cas est différent.

Beaucoup de méthodes ont été proposées pour calculer le modèle géométrique direct des robots manipulateurs, telles que : la convention de *Denavit-Hartenberg (DH)* et celle de *Khalil-Kleinfinger (KK)*.

Le principe de toutes ces méthodes est d'associer à chaque corps un repère (au début ou à la fin du corps selon la méthode choisie), et de calculer les matrices de transformation entre chaque deux repères successifs, et la matrice de transformation totale est le produit de toutes ces matrices.

On choisit la convention de DH, le modèle géométrique direct du robot planaire de La figure 1.2 est calculé comme suit :

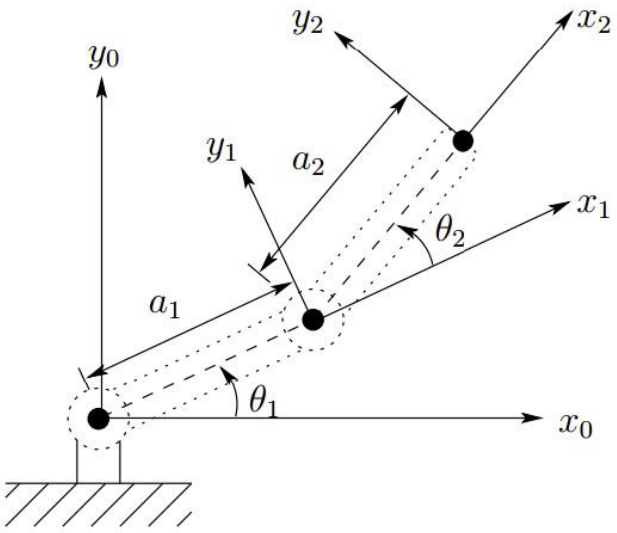

FIGURE 1.2 – Robot manipulateur planaire à deux corps.

– Première chose est de construire la table des paramètres selon la convention :

TABLE 1.1 – Les paramètres des articulations du robot planaire.

Articulation	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2

– Deuxième chose est la construction du modèle géométrique en utilisant la table 1.1 :

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1c_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.11}$$

$$T_2^1 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2c_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

$$T_2^0 = T_1^0\, T_2^1 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1c_1 + a_2c_{12} \\ s_{12} & c_{12} & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.13}$$

tel que $c_1 = \cos(\theta_1)$ et $s_1 = \sin(\theta_1)$ et $c_{12} = \cos(\theta_1 + \theta_2)$ et $s_{12} = \sin(\theta_1 + \theta_2)$

Le modèle géométrique inverse est généralement plus compliqué que le modèle géométrique direct, par ce qu'on doit trouver les valeurs de n inconnus à partir d'une seule matrice 4×4.

$$T_n^0(q_1, q_2, \ldots, q_n) = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix} \in SE(3) \tag{1.14}$$

On peut distinguer deux méthodes de calcul du modèle géométrique inverse : méthode de *newton* et celle de *Pieper* [7].

1.4 Modèle cinématique direct et inverse

Un repère mobile possède deux vitesses : linéaire et angulaire.

La vitesse linéaire est associée à un point mobile, tandis que la vitesse angulaire est associée à un repère rotatif. Ainsi, la vitesse linéaire d'un repère mobile est simplement

la vitesse de son origine. La vitesse angulaire pour un repère mobile est liée à la dérivée temporelle de la matrice de rotation qui décrit l'orientation instantanée du repère.

L'idée utilisée pour calculer la dérivée d'une matrice est de multiplier cette matrice par une matrice spéciale que l'on appelle la matrice *jacobéenne* $J(q)$ [6].

$$\dot{x} = J(q)\,\dot{q} \tag{1.15}$$

$$\dot{q} = J^{-1}(q)\,\dot{x} \quad \text{si} \quad J(n \times n) \text{ est inversible} \tag{1.16}$$

L'objectif du modèle cinématique direct est de décrire les vitesses des coordonnées opérationnelles en fonction des vitesses articulaires, et l'objectif du modèle cinématique inverse est de calculer à partir d'une configuration donnée les vitesses articulaires qui assurent une vitesse opérationnelle imposée et pour ce faire on a besoin de calculer J^{-1}, ce qui pose quelques problèmes, tels que : *la singularité*, et *la redondance*.

L'une des méthodes de calcul de la matrice jacobéenne est la méthode de propagation des vitesses, qui consiste à calculer les vitesses linéaires et angulaires en fonction des vitesses articulaires, et ce, en commençant de la base vers l'effecteur [5].

Si la matrice jacobéenne est carrée et non singulière, son inverse est calculé simplement, mais pour le cas où J est n'est pas carrée (un degré de liberté de plus), on parle alors de la redondance, il y a alors possibilité de plusieurs solutions. Quelques méthodes sont proposées pour traiter ce problème, telles que : la pseudo-inverse avec un terme d'optimisation, les vecteurs de coordonnées supplémentaires. Et pour le cas où J est carrée mais singulière, une solution consiste à augmenter le nombre de degrés de liberté du mécanisme [8].

1.5 Planification de la trajectoire et planification du chemin

La planification du chemin fournit une description géométrique du mouvement du robot, mais elle ne précise pas les aspects dynamiques du mouvement. Par exemple, quelles devraient être les vitesses et les accélérations articulaires tout en traversant le chemin ? Ces questions sont traitées par un planificateur de trajectoire. Le planificateur de trajectoire

calcule une fonction $q^d(t)$ de class C^2 qui spécifie complètement le mouvement du robot pendant qu'il traverse le chemin.

1.5.1 Espace des configurations et espace de travail

Pour planifier le mouvement d'un robot nous devons être capable de spécifier sa position et son orientation et plus précisément, nous devons être capable de spécifier la position de chaque point du robot, et d'assurer qu'elle n'est pas en collision avec un obstacle [9]. Alors quelles sont toutes les informations nécessaires pour spécifier la position de chaque point du robot ? et comment le robot doit tenir compte des obstacles à éviter ?

Pour répondre à ces questions nous allons définir quelques notations :

- **L'espace de travail $\mathcal{W}$**

  L'espace de travail $\mathcal{W}$ permet de définir l'ensemble des positions atteignables par le mobile $\mathcal{M}$; $\mathcal{W}$ peut contenir un ensemble de n obstacles ,ce qui peut être exprimé par la notation $\mathcal{WO}$; $\mathcal{M}$ est physiquement présent dans $\mathcal{W}$. Une position est atteignable si $\mathcal{M}$ n'est pas en collision avec un obstacle ; si $\mathcal{W}$ est un espace $2D$, $\mathcal{M}$ agit dans le plan ; si $\mathcal{W}$ est un espace $3D$, $\mathcal{M}$ agit dans l'espace. Dans le cas d'un mobile non ponctuel, les mouvements de $\mathcal{M}$ dans $\mathcal{W}$ sont définissables par comparaison entre la taille de $\mathcal{M}$ et celle des espaces libres entre les obstacles, c-à-dire dans $\mathcal{W}_{free}$ qui exprimer par l'équation suivant :

  $$\mathcal{W}_{free} = \mathcal{W} \setminus \bigcup_1^n \mathcal{WO}_i \tag{1.17}$$

  tel que $\setminus$ est l'opérateur de soustraction [9].

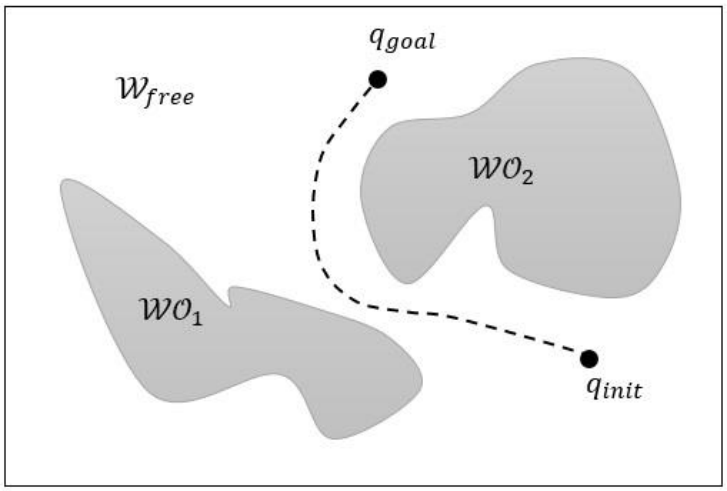

FIGURE 1.3 – L'espace de travail et les obstacles, et le chemin entre q_{init} et q_{goal}.

- **L'espace des configurations $\mathcal{C}$**

  Pour uniformiser la description de l'espace de travail, $\mathcal{W}$ est transformé *en Espace des configurations*. Cette transformation en espace des configurations permet de substituer la recherche d'une trajectoire de $\mathcal{M}$ dans $\mathcal{W}$ par la recherche de la trajectoire d'un point dans $\mathcal{C}$. $\mathcal{M}$ étant un robot à n degrés de liberté, une configuration est définie par un vecteur $q = (q_1, q_2, \ldots, q_n)$ composé de l'ensemble des n valeurs des degrés de liberté de $\mathcal{M}$. $\mathcal{C}$ est défini par l'ensemble des valeurs possibles des n variables de q. Les problèmes de planification de mouvement dans $\mathcal{W}$ sont isomorphes aux problèmes de recherche dans un espace de dimension n. Alors L'espace des configurations $\mathcal{C}$ se divise en deux sous-ensembles complémentaires : $\mathcal{C}_{free}$, et $\mathcal{C}_{obs}$ qui sont exprimés par la relation :

$$\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_1^n \mathcal{C}_{obs} \tag{1.18}$$

  Tel que $\mathcal{C}_{free}$ est l'ensemble des configurations dites libres (en dehors des obstacles et sans contact). Et $\mathcal{C}_{obs}$ est l'ensemble des configurations en collision (en collision ou en contact) avec les obstacles. Les images des obstacles de $\mathcal{W}$ dans $\mathcal{C}$ sont appelées $\mathcal{C}$-obstacles [9].

- **Le problème de planification du chemin**

  Pour faire déplacer le robot d'un point à l'autre, il faut trouver une succession de configurations admissibles lui permettant d'atteindre son objectif. Le problème de

planification de chemin correspond à la recherche d'une séquence de configurations appartenant à $\mathcal{C}_{free}$ depuis un état de départ q_{init} jusqu'à un état d'arrivée q_{goal} [9]. La solution dans l'espace des configurations est la fonction $path(s)$ qui est continue et de classe $\mathcal{C}^0$, tel que :

$$path(s) : [0,1] \rightarrow \mathcal{C}_{free} \tag{1.19}$$

avec

$$path(0) = q_{init}$$
$$path(1) = q_{goal}$$

1.5.2 Les méthodes de planification de mouvement

Il existe deux principales catégories de méthodes pour la planification de mouvement :

- La première est composée de méthodes dites *déterministes*, appelées ainsi car elles permettent de retrouver le même chemin à chaque exécution, sous réserve d'avoir des conditions initiales équivalentes. Les méthodes déterministes sont dites *complètes en résolution*. Quelques algorithmes bien connus : champs de potentiel, décomposition cellulaire, diagrammes de Voronoï ...
- La seconde catégorie est composée des méthodes *probabilistes*. Ces méthodes ne sont pas complètes en résolution, mais elles garantissent de trouver une solution s'il en existe une. On dit qu'elles sont *probabilistiquement complètes*. Ces méthodes ne trouveront pas forcément le même chemin à chaque exécution, même avec des conditions initiales similaires.

Definition 1.5.1. [10] *une méthode de planification est dite complète en résolution si elle peut générer un chemin continu* $path(s) : [0,1] \rightarrow \mathcal{C}_{free}$ *s'il existe, ou elle peut signaler l'échec si aucun chemin existe, dans un intervalle de temps fini.*

Definition 1.5.2. [10] *une méthode de planification est dite probabilistiquement complète si elle peut trouver un chemin dans un intervalle de temps fini avec une forte probabilité, quand il en existe un.*

Nous présentons ici deux méthodes déterministes bien connues (la décomposition en cellules et champs de potentiel) et une introduction sur les méthodes probabilistes

1.5.2.1 Méthodes par décomposition exact en cellules

Ces structures représentent l'espace libre par l'union des régions simples appelées les cellules. Les frontières partagées des cellules ont souvent une signification physique telle qu'un changement de l'obstacle le plus étroit ou un changement de champ de vision aux obstacles environnants. Deux cellules sont adjacentes si elles partagent une frontière commune. Un graphe d'adjacence code les relations d'adjacence des cellules, où un noeud correspond à une cellule et un arc relie entre les noeuds des cellules adjacentes.

Si la décomposition est faite, la planification du chemin avec la décomposition en cellule se fait généralement en deux étapes : d'abord, le planificateur détermine les cellules qui contiennent le départ et la destination, puis le planificateur recherche un chemin dans le graphe d'adjacence [9].

Sous cette technique, beaucoup de méthodes sont proposées, telles que la décomposition trapézoïdale, *la décomposition de Morse* [11], *la décomposition basée sur la visibilité* [12].

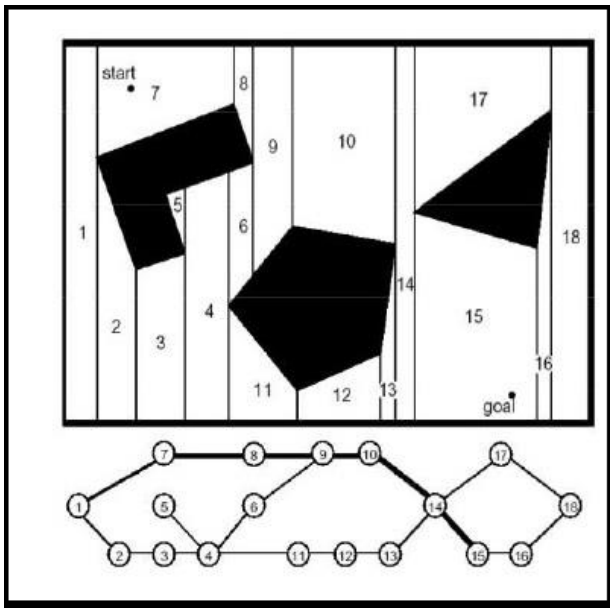

FIGURE 1.4 – La décomposition trapézoïdale et sa représentation en graphe.

1.5.2.2 Méthodes de champ de potentiel

Les méthodes de champs de potentiel sont initialement proposées par Khatib O. [13]. Elles consistent à considérer le robot mobile comme une particule soumise à divers champs électromagnétiques régissant son mouvement. Ces champs de potentiel sont de deux types :

- Un champ de potentiel attractif $U_{att}(q)$ provenant de la position finale du système à atteindre.
- Un champ de potentiel répulsif $U_{rep}(q)$ provenant des obstacles statiques et mobiles de l'environnement.

Et le champ résultant qui agit sur le robot est la somme du champ attractif et du champ répulsif.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{1.20}$$

1.5.2.3 Méthodes probabilistes

Les méthodes déterministes se fondent sur une représentation explicite de la géométrie de $\mathcal{C}_{free}$, et quand l'espace des configurations est fait de plusieurs dimensions (plus de cinq), les méthodes déterministes deviennent impraticables. Ce cas peut être résolu avec les méthodes probabilistes, de sorte que ces méthodes utilisent plusieurs stratégies pour générer des échantillons dans $\mathcal{C}_{free}$ (configurations sans collision) et elles font des connexions (des chemins) entre ces échantillons pour obtenir des solutions aux problèmes de planification de chemin. Les méthodes probabilistes sont généralement classées en deux grandes familles :

- Les méthodes à *requêtes multiples* : i.e. *carte des routes probabilistes* [2] (PRM) pour (Probabilistics Roadmaps). Ces méthodes sont constituées de deux phases : *la phase d'apprentissage* et *la phase de traitement des requêtes*, la première phase consiste à générer des échantillons dans l'espace $\mathcal{C}_{free}$ et de faire toutes les connexions possibles et les sauvegarder dans un graphe. Une fois que le graphe est construit, la deuxième phase sera prête pour répondre à des requêtes de planification de chemin. Ces méthodes peuvent trouver le chemin entre les différentes configurations de départ et d'arrivée en utilisant la même carte. Plus de détails à ce propos seront discutés dans le chapitre 3.

– Les méthodes à *requête simple* : Dans ce cas l'exploration se fait en construisant de manière incrémentale des arbres de recherche (des graphes ne contenant aucun cycle) à partir de la configuration initiale vers la configuration finale. Ces techniques, contrairement aux précédentes, n'explorent pas tout l'espace $\mathcal{C}_{free}$ mais explorent seulement certaines de ses régions afin de trouver la configuration finale, pour minimiser le temps de réponse à la requête. Plusieurs techniques d'exploration sont proposées. C'est le cas de *l'arbre d'exploration rapide (Rapidly-exploring Random Tree RRT), l'arbre des espaces expansifs (Expansive-Spaces Tree EST).*

1.6 Conclusion

Nous avons présenté dans ce chapitre une base sur la modélisation des robots et la planification de la trajectoire qui nous aidions à réaliser un planificateur de chemin basé sur la carte de route probabiliste.

Chapitre 2

Théorie des graphes et test de collision

2.1 Théorie des graphes

L'histoire de *la théorie des graphes* début peut-être avec les travaux d'Euler au $XVIII^{\text{ème}}$ siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg (Figure 2.1), les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ.

La théorie des graphes s'est alors développée dans diverses disciplines telles que la chimie, la biologie, les sciences sociales. Depuis le début du $XX^{\text{ème}}$ iéme siècle, elle constitue une branche à part entière des mathématiques, grâce aux travaux de Königsberg, Menger, Cayley puis de Berge et d'Erdös.

De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques,...

Les graphes constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs. Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance qu'y revêt l'aspect algorithmique.

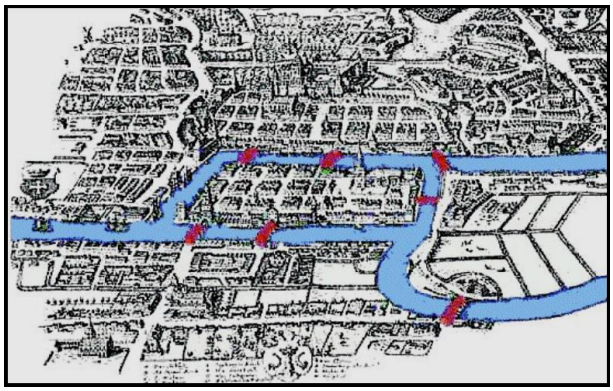

FIGURE 2.1 – Problème des ponts de Königsberg.

Definition 2.1.1. [14] *Un graphe G est un couple (V, E) où :*

- *$V = (v_1, v_2, \ldots, v_n)$ Est l'ensemble fini dont les éléments sont appelés sommets ou noeuds.*
- *$E = (e_1, e_2, \ldots, e_m)$ Est l'ensemble fini du produit cartésien $V \times V$ dont les éléments sont appelés arcs ou arête.*

2.1.1 Graphe orienté

Dans beaucoup d'applications, les relations entre éléments d'un ensemble sont orientées, i.e., un élément x peut être en relation avec un autre y sans que y soit nécessairement en relation avec x. On parle alors de graphe orienté (en Anglais *directed graph* ou plus simplement *digraph*).

Alors, Pour un arc $e = (v_i, v_j)$, v_i est l'extrémité initiale, v_j l'extrémité finale (ou bien origine et destination). L'arc e part de v_i et arrive à v_j. Si l'arc e part de v_i et arrive au même sommet v_i on l'appelle *une boucle* (Figure 2.2).

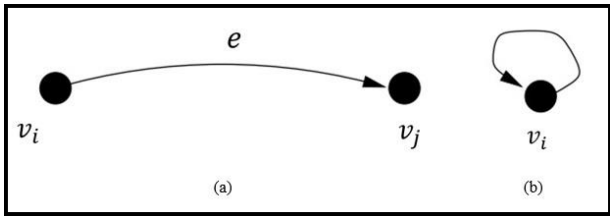

FIGURE 2.2 – (a) Un arc e. (b) Une boucle.

Un p-graphe est un graphe dans lequel il n'existe jamais plus de p arcs de la forme (i, j) entre deux sommets quelconques.

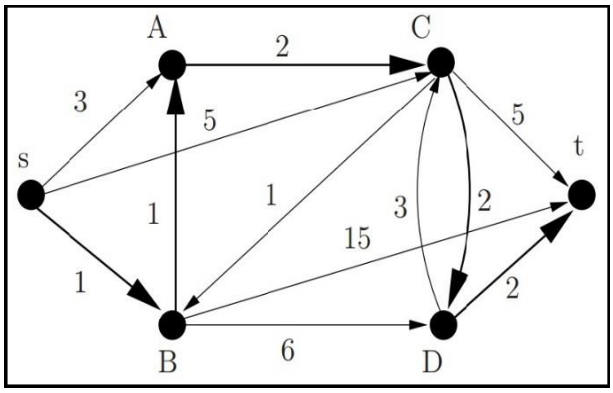

FIGURE 2.3 – Graphe orienté

2.1.2 Graphe non orienté

Lors de l'étude de certaines propriétés, il arrive que l'orientation des arcs ne joue aucun rôle. On s'intéresse simplement à l'existence d'arc(s) entre deux sommets (sans en préciser l'ordre).

Un arc sans orientation est appelé *arête*. V est constitué non pas de couples, mais de paires de sommets non ordonnés. Pour une arête (v_i, v_j), on dit que e est *incidente* aux sommets $v_i$etv_j.

Un multigraphe $G = (V, E)$ est un graphe pour lequel il peut exister plusieurs arêtes entre deux sommets.

Un graphe $G = (V, E)$ est simple :

- S'il n'est pas un multigraphe ;
- S'il n'existe pas de boucles.

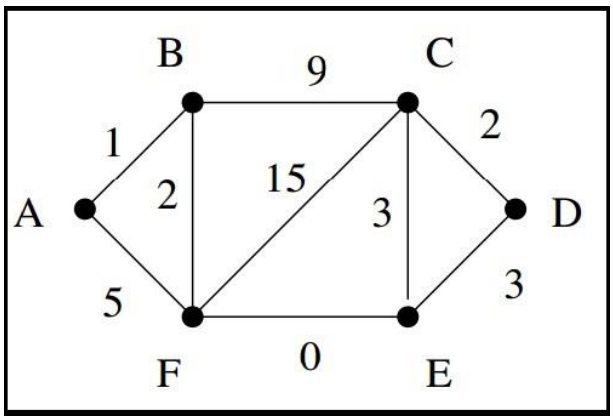

FIGURE 2.4 – Graphe non orienté

2.1.3 Quelques principales Définitions

- *Adjacence*
  - Deux sommets sont adjacents s'ils sont joints par un arc.
  - Deux arcs sont adjacents s'ils ont au moins une extrémité commune.
- Degrés
  - Le demi-degré extérieur de v_i, $d^+(v_i)$, est le nombre d'arcs ayant v_i comme extrémité initiale.
  - Le demi-degré intérieur de v_i, $d^-(v_i)$, est le nombre d'arcs ayant v_i comme extrémité finale.
  - Le degré de v_i est $d(v_i) = d^+(v_i) + d^-(v_i)$. Le degré d'un sommet d'un graphe non orienté est le nombre d'arêtes qui lui sont incidentes.
- Un graphe est *symétrique* si, pour tout arc $e_1 = (v_i, v_j)$ appartenant à E, l'arc $e_2 = (v_j, v_i)$ appartient également à E.
- Un graphe est *antisymétrique* si, pour tout arc $e_1 = (v_i, v_j)$ appartenant à E, l'arc $e_2 = (v_j, v_i)$ n'appartient pas à E.
- Un graphe $G = (V, E)$ est dit complet si, pour toute paire de sommets (v_i, v_j), il existe au moins un arc de la forme (v_i, v_j) ou (v_j, v_i) [14].

2.1.4 Représentations d'un graphe

Un certain nombre de représentations existent pour décrire un graphe. En particulier, elles ne sont pas équivalentes du point de vue de l'efficacité des algorithmes [14]. On distingue principalement trois représentations :

- La représentation par matrice d'adjacence.
- La représentation par matrice d'incidence sommets-arcs (ou sommets-arêtes dans le cas non orienté).
- La représentation par listes d'adjacence.

Dans notre mémoire, nous nous intéressons à la représentation par matrice d'adjacence.

2.1.5 Représentation d'un graphe par matrice d'adjacence

On considère un 1-graphe. La matrice d'adjacence fait correspondre les sommets origine des arcs (placés en ligne dans la matrice) aux sommets destination (placés en colonne). Dans le formalisme matrice booléenne, l'existence d'un arc (v_i, v_j) se traduit par la présence d'un 1 à l'intersection de la ligne v_i et de la colonne v_j ; l'absence d'arc par la présence d'un 0 (dans un formalisme dit matrice aux arcs les éléments représentent le nom de l'arc ou le poids i.e. la distance).

Un graphe orienté quelconque a une matrice d'adjacence quelconque, alors qu'un graphe non orienté possède une matrice d'adjacence symétrique. L'absence de boucle se traduit par une diagonale nulle.

La matrice d'adjacence du graphe de La figure 2.3 est la suivante :

		Destinations					
		s	A	B	C	D	t
Origines	s	0	3	1	5	0	0
	A	0	0	0	2	0	0
	B	0	1	0	1	6	15
	C	0	0	1	0	2	5
	D	0	0	0	3	0	2
	t	0	0	0	0	0	0

TABLE 2.1 – La matrice d'adjacence

2.1.6 Problème du plus court chemin

Les problèmes de cheminement dans les graphes (en particulier la recherche d'un plus court chemin) comptent parmi les problèmes les plus anciens de la théorie des graphes et les plus importants par leurs applications.

Definition 2.1.2. [14] *Soit $G = (V, E)$ un graphe valué, on associe à chaque arc $e = (v_i, v_j)$ une longueur $l(e)$. Le problème du plus court chemin entre v_i et v_j est de trouver*

un chemin $\mu(v_i, v_j)$ *de* v_i *à* v_j *tel que :*

$$l(\mu) = \sum_{u \in \mu} l(u) \text{ soit minimale} \tag{2.1}$$

Plusieurs algorithmes sont proposées pour résolue ce problème, ces algorithmes sont différents suivant les propriétés des graphes ($l(e) \geq 0, \forall e \in E$, G sans circuit,...) et suivant le problème considérée (recherche du plus court chemin d'un sommet vers tous les autres, ou bien entre deux sommets,...).

Nous présentons deux algorithmes pour calculer le plus court chemin : *Dijkstra* et A^*, que nous avons utilisés dans notre réalisation.

2.1.6.1 Algorithme de Dijkstra

Edgser Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres [15]. Son principe de fonctionnement est comme suit :

Numérotons les sommets du graphe $G = (V, E)$ de 1 à n. Supposons que l'on s'intéresse aux chemins partant du sommet i. On construit un vecteur $\lambda = (\lambda(1), \lambda(2), \ldots, \lambda(n))$ ayant n composantes tel que $\lambda(j)$ soit égal à la longueur du plus court chemin allant de i au sommet j. On initialise ce vecteur à $c_{i,j}$, c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous :

$$c_{i,j} = \begin{cases} 0 & \text{si } i = j \\ \infty & \text{si } i \neq j \text{ et } (i,j) \notin E \\ \delta(i,j) & \text{si } i \neq j \text{ et } (i,j) \in E \end{cases} \tag{2.2}$$

Où $\delta(i, j)$ est le poids (la longueur) de l'arc (i, j). Les $c_{i,j}$ doivent être strictement positifs. On construit un autre vecteur p pour mémoriser le chemin pour aller du sommet i au sommet voulu. La valeur $p(i)$ donne le sommet qui précède i dans le chemin.

On considère ensuite deux ensembles de sommets, S (visités) initialisé à $\{i\}$ et T (non visités) initialisé à $\{1, 2, 3, \ldots, n\} - i$. A chaque itération de l'algorithme, on ajoute à S un sommet jusqu'à ce que $S = V$ de telle sorte que le vecteur λ donne à chaque étape le coût minimal des chemins de i aux sommets de S.

Algorithm 1 [14] Algorithme de Dijikstra

Entrées: $G(V, E)$

Pour $j \leftarrow 1$ n **Faire**
    $\lambda(j) = c_{i,j}$, $p(j) = NIL$;
Fin Pour
S={ } ; T={1,2,3,. . . ,n} ;
Tant que T n'est pas vide **Faire**
    Choisir i dans T tel que $\lambda(i)$ est minimum ;
    Retirer i de T et l'ajouter à S ;
    Pour chaque adjacent j de i, avec j dans T **Faire**
        Si $\lambda(j) > \lambda(i) + \sigma(i, j)$ **Alors**
            $\lambda(j) \leftarrow \lambda(i) + \sigma(i, j)$;
            $p(j) \leftarrow i$;
        Fin Si
    Fin Pour
Fin Tant que

2.1.6.2 Algorithme de A^*

Cet algorithme a été proposé pour la première fois par *Peter E. Hart, Nils John Nilsson et Bertram Raphael* en 1968 [16] à l'institut de recherche de Standford. Il s'agit d'une extension de l'algorithme de Dijkstra de 1959. L'idée de cet algorithme est d'utilisée l'heuristique pour guidé la recherche. L'étoile signifié que sa recherche dans un graphe au sommet désirer fait sous forme un arbre guidée, qui a la forme d'une étoile. Son principe de fonctionnement est comme suit :

Au début on donne à chaque sommet dans le graphe $G(V, E)$ une valeur heuristique $h(n)$: l'estimation du coût entre ce sommet et le sommet désirer (dans la planification du chemin on peut utiliser la distance euclidienne ou autres distances), ensuit à chaque itération la méthode choisie le chemin qui minimise le critère :

$$f(n) = g(n) + h(n) \tag{2.3}$$

Tel que $g(n)$ est le coût entre le sommet courant et le sommet adjacent.

L'algorithme A^* utilise trois listes : Liste_ouverte (contient les adjacents non visités), Liste_fermée (contient les sommets visités), Liste_provenance (contient les sommets-origines de chaque adjacent visité, utilisés pour construire le chemin).

A chaque itération on prend de la Liste_ouverte le sommet qui a le$f$ minimum que on appelle Courant et on l'efface de la Liste_ouverte et on l'ajoute à la Liste_fermée ; ensuit, si le sommet Courant n'existe pas dans la Liste_fermée, alors pour chaque adjacent de Courant, on calcule son f et on l'ajoute à la Liste_ouverte et on ajoute le sommet Courant à la Liste_provenance si cet adjacent n'existe pas dans la Liste_ouverte ; et si cet adjacent existe dans la Liste_ouverte, et si son nouveau g inférieure à son g ancienne alors on change son g et on change son origine dans la Liste_provenance à ce nouvelle Courant. Jusqu'à ce que la Liste_ouverte soit vide.

Le A^* produit un chemin optimale si la fonction heuristique est *admissible* : ne surestime jamais le coût exact $\widehat{h}(n)<h(n)$ [16].

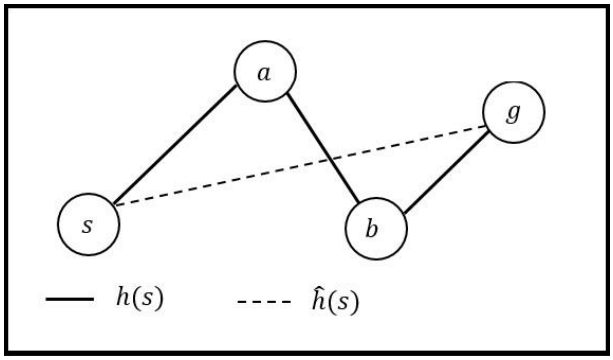

FIGURE 2.5 – La fonction heuristique est admissible si $\widehat{h}(n)<h(n)$.

Et dans le cas de la recherche dans un graphe, il faut aussi que la fonction heuristique soit *monotone* : pour chaque deux sommets n et m du graphe, la valeur heuristique de n est inférieur ou égale à la valeur heuristique de m plus le coût entre eux [16].

$$h(n) \leqslant h(m) + d(n,m) \tag{2.4}$$

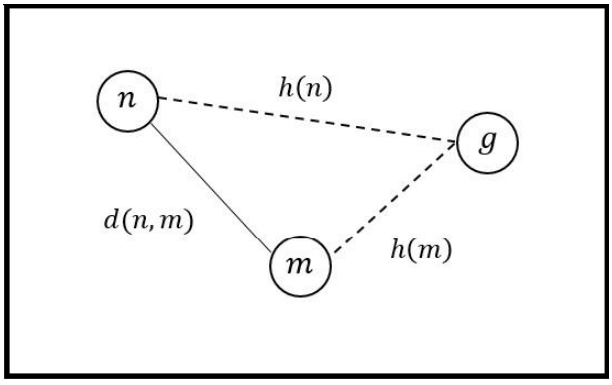

FIGURE 2.6 – La fonction h est monotone si $h(n) \leqslant h(m) + d(n,m)$.

Dans l'algorithme suivant, on suppose que la fonction heuristique est monotone.

Algorithm 2 Pseudo-code de l'algorithme A^*

Entrées: $G(V, E)$, S_init, S_désiré

Sorties : Chemin

```
1: H := Les valeurs heuristiques de tous les sommets ;
2: Liste_ouverte := {S_init} ;
3: Liste_fermée := {S_init} ;
4: g := {0} ;
5: f := {H(S_init)} ;
6: Liste_provenance := {} ; Noeuds := {} ;
7: Tant que la Liste_ouverte n'est pas vide Faire
8:     Courant := Le sommet dans Liste_ouverte qui a le f minimum ;
9:     Pour chaque adjacent de courant Faire
10:        Si l'adjacent est dans la Liste_fermée Alors
11:            Continue ;
12:        Fin Si
13:        Dist := E(Courant, adjacent) + g[Courant] ;
14:        Si l'adjacent est dans Liste_ouverte Alors
15:            Si Dist < g[adjacent] Alors
16:                g[adjacent]:= Dist ;
17:                f[adjacent]:= g[adjacent] + H[adjacent] ;
18:                Liste_provenance [adjacent] := Courant ;
19:                Continue ;
20:            Si non
21:                Continue ;
22:            Fin Si
23:        Fin Si
24:        Ajouter l'adjacent à Liste_ouverte ;
25:        Ajouter E(Courant,adjacent) + g[Courant] à g ;
26:        Ajouter g[dernière] + H[adjacent] à f ;
```

```
27:         Ajouter Courant à la Liste_provenance et ajouter l'adjacent à Noeuds ;
28:         Si Noeuds[dernière] est S_désiré Alors
29:             Returner Chemin := reconstruitChemin(Liste_provenance, Noeuds) ;
30:         Fin Si
31:     Fin Pour
32:     Enlever Liste_ouverte[Courant] ;
33:     Ajouter Courant à la Liste_fermée ;
34:     Enlever g[Courant] ;
35:     Enlever f[Courant] ;
36: Fin Tant que
37: Returner Pas de chemin ;

38: Fonction RECONSTRUITCHEMIN(Liste_provenance, Noeuds)
39:     Chemin := {S_dsir} ;
40:     Tant que Chemin[dernière] ≠ S_init Faire
41:         k := l'indice de Chemin[dernière] dans Noeuds ;
42:         ajouter Liste_provenance [k] à Chemin ;
43:     Fin Tant que
44:     Returner Chemin
45: Fin Fonction
```

2.2 Test de collision

Le test de collision est une composante essentielle de la planification fondée sur l'échantillonnage. Même si elle est souvent traitée comme une boîte noire, il est important d'étudier son fonctionnement interne pour comprendre les informations qu'il fournit et son coût de calcul associé. Dans la plupart des applications de planification de mouvement, la majorité du temps de calcul est consacré au test de collision [17].

Dans cette deuxième partie de ce chapitre, nous allons présenter quelques méthodes de test de collision pour un environnement en 2D, et les techniques proposées dans la littérature pour optimiser ces temps de calcule.

2.2.1 Problème du point dans le polygone

On peut classer les polygones en deux catégories : convexes et non convexes. On dit qu'un polygone est convexe si le segment qui relier entre deux points quelconques appartenant à ce polygone reste toujours à l'intérieur de ce dernier. Le teste d'appartenance d'un point à un polygone convexe est facile si on connut les équations de ces frontières, en remplaçant les coordonnés de ce point dans les inégalités des demi-plans qui forment le polygone [18].

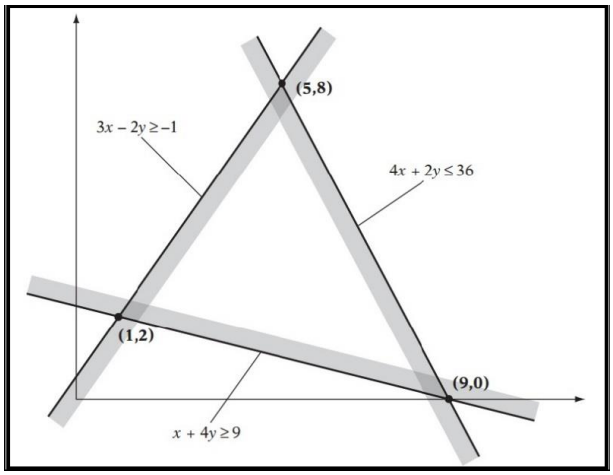

FIGURE 2.7 – Un polygone convexe peut être défini par l'intersection des demi-plans, ici le triangle (5,8),(1,2),(9,0) est l'intersection de $3x-2y \geq -1$ et $4x+2y \leq 36$ et $x+4y \geq 9$.

Mais si le polygone est non convexe le concept des demi-plans n'est plus utilisé. Deux méthodes communes se trouvent dans la littérature pour résolu ce problème : la méthode de pair-impaire (even-odd or parity rule) et l'indice du point (winding number).

2.2.1.1 Principe de la méthode pair-impaire

Soit R un point appartenant à un polygone $P = P_1P_2 \dots P_n$, on trace à partir du point R une droite vers un point quelconque S qui est en d'hors de polygone, cette droite RS doit couper les bordures $e_i = P_i\,P_{i+1}$ du polygone n fois (Figure 2.8), si n est pair alors le point n'est pas dans le polygone, et si n est impaire alors le point est dans le polygone [19]. On peut facilement traduire cette définition en algorithme de test. Mais cette méthode n'est pas valable si R est un sommet ou appartient à la bordure du polygone, ou encore

si la droite passe par un sommet.

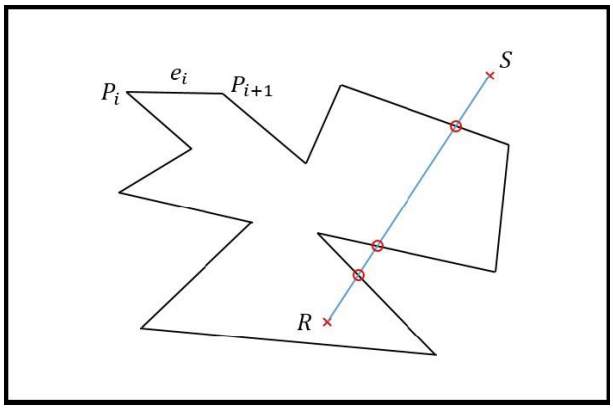

FIGURE 2.8 – Dans ce cas $n = 3$, alors R est dans P.

2.2.1.2 Principe de la méthode de l'indice de point

L'indice ω d'un point par rapport à un lacet $C(t)$ est le nombre de tours au sens trigonométrique réalisé par un lacet autour d'un point, si ce nombre est différent de zéro alors le point est dans lacet. L'indice d'un point par rapport à un lacet est donné par la formule :

$$\omega(R, C) = \frac{1}{2\pi} \int_a^b \mathrm{d}\varphi(t) \tag{2.5}$$

En utilisant ce principe, on considère que le polygone est un lacet discrétisé c'est-à-dire est un ensemble de segments qui forment le polygone [19]. Alors l'indice du point R est la somme des angles signés φ_i entre $\overline{RP_i}$ et $\overline{RP_{i+1}}$.

$$\omega(R, P) = \frac{1}{2\pi} \sum_{i=0}^{n-1} \varphi_i \tag{2.6}$$

Tel que R est le point à tester s'il est dans le polygone P, et n est le nombre des sommets de P.

$$\omega(R, P) = \frac{1}{2\pi} \sum_{i=0}^{n-1} \arccos \frac{\langle P_i \mid P_{i+1} \rangle}{||P_i|| ||P_{i+1}||} \; sign \begin{vmatrix} P_i^x & P_{i+1}^x \\ P_i^y & P_{i+1}^y \end{vmatrix} \tag{2.7}$$

La formule (2.7) peut utilisée dans l'algorithme de test si un point est dans un polygone. Mais, lorsque cet algorithme de test est appelé plusieurs fois dans la planification du chemin, le calcule de *arccos* et la racine carrée à chaque test rend l'algorithme lourd. Pour cela, K.Hormann et A.Agathos proposent dans [19] plusieurs simplification pour éviter ces deux fonctions : Classification des sommets par quadrants, Rayon-croisement.

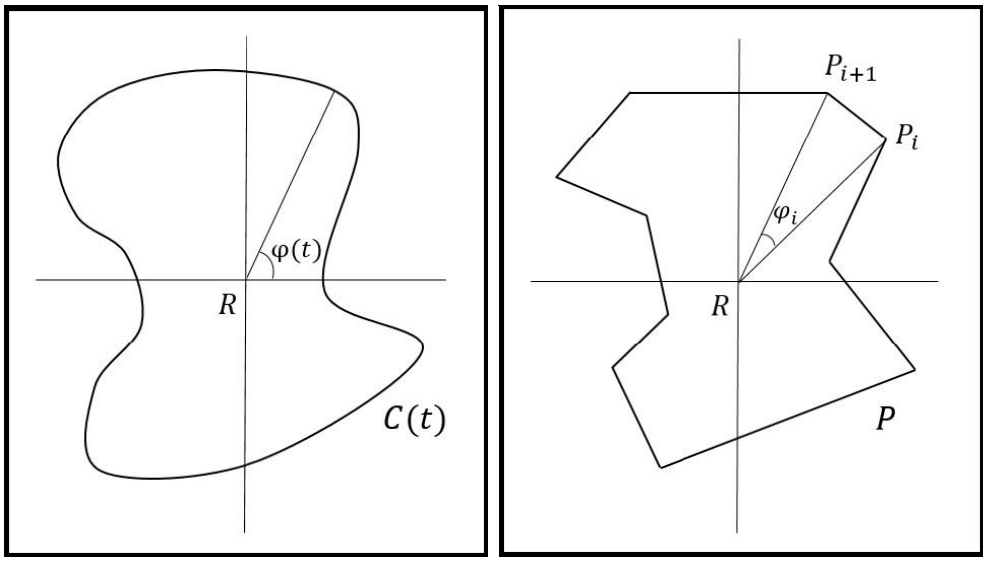

FIGURE 2.9 – Le polygone P est un lacet discrétisé.

2.2.2 Test d'intersection d'un segment de droite avec un polygone

Pour tester l'intersection d'un segment de droite $\overline{S_1S_2}$ avec un polygone P il suffit de tester l'intersection de $\overline{S_1S_2}$ avec chaque segment $\overline{P_iP_{i+1}}$ du polygone P.

$$\overline{S_1S_2} = \{y_s = a_s x + b_s \; ; \; y_s \in [y_{s_1}, y_{s_2}] \; et \; x_s \in [x_{s_1}, x_{s_2}]\} \tag{2.8}$$

$$\overline{P_iP_{i+1}} = \{y_p = a_p x + b_p \; ; \; y_p \in [y_{p_i}, y_{p_{i+1}}] \; et \; x_p \in [x_{p_i}, x_{p_{i+1}}]\} \tag{2.9}$$

L'intersection est calculée par l'égalisation de deux équations, et on a 3 cas possible :

$$\overline{S_1S_2} \cup \overline{P_iP_{i+1}} = \begin{cases} \text{un point } (x, y) \\ \text{un segment de droite } \overline{S_1S_2} = \overline{P_iP_{i+1}} \\ \emptyset \end{cases} \tag{2.10}$$

Dans le cas de la planification de chemin on n'a pas besoin de trouver l'intersection, qu'elle soit un point ou un segment, mais on n'a besoin que à d'un test logique : voir s'il y a intersection ou non, alors l'expression (2.10) devient :

$$\overline{S_1S_2} \cup \overline{P_iP_{i+1}} = \begin{cases} 1 & \text{si l'intersection est un point ou un segment} \\ 0 & \text{si l'intersection est } \emptyset \end{cases} \tag{2.11}$$

2.2.3 Test d'intersection de deux polygones

On dite que deux polygones P_1 et P_2 sont pas en collision s'il n'y a pas d'intersection entre chaque segment du polygone P_1 et les segments du polygone P_2 ; mais ce test prend $(n \times m)$ temps [20] (n, m : nombre des segments de deux polygone P_1 et P_2 respectivement). Pour cela plusieurs méthodes sont proposées pour optimiser le temps du test tel que : la décomposition en triangle [20], utilisation d'heuristique [21],etc.

2.3 Conclusion

Nous avons présenté dans ce chapitre deux outil très important pour réaliser un planificateur de chemin par carte de route probabiliste, tel que : la théorie des graphes qui nous aidions à trouver le plus court chemin, et le test de collision qui nous permettons d'éliminer les configurations qui sont en collision.

Chapitre 3

La carte de route probabiliste (PRM)

Plus récemment, plusieurs approches non déterministes sont apparues, elles ne reposent plus sur une construction préalable d'une représentation exacte ou approchée de l'espace de configurations. Certaines de ces méthodes se basent sur l'exploration de l'espace de configurations admissibles, d'autres visent à capturer sa connectivité. Ces nouvelles méthodes de planification dites méthodes probabilistes, permettent de remédier à la complexité exponentielle du problème. Par ailleurs, elles vérifient une propriété de complétude probabiliste. Certes cette propriété est moins forte que la complétude déterministe vérifiée par les méthodes exactes, mais ces nouvelles approches sont désormais moins sensibles à la dimension de l'espace de recherche et s'avèrent très efficaces en pratique.

Les méthodes probabilistes ont été introduites en parallèle par Kavraki et Latombe sous le nom de PRM (Probabilistic Roadmap Planner) [22][2] et Svestka et Overmars sous le nom de PPP (Probabilistic Path Planner) [3]. Depuis, plusieurs variantes de ces méthodes ont été proposées dans la littérature mais toutes se basent sur le même concept.

On présente dans ce chapitre de manière détaillée les méthodes qui se basent sur la carte de route probabiliste PRM ; puis, nous allons présenter trois méthodes d'échantillonnages : l'échantillonnage uniforme qui est utilisé dans les méthodes probabilistes classiques, et l'échantillonnage gaussien et l'échantillonnage par pont qui traitent le problème des passages étroits ; ensuite, nous allons présenter deux planificateurs locaux : planificateur de ligne droite (Straight-line planner), et le planificateur de rotation-à-s (Rotate-at-s) ; enfin, nous présenterons trois formules pour calculer la distance entre deux configurations dans $\mathcal{C}$.

3.1 La structure générale d'un planificateur de PRM

Avant de présenter les algorithmes d'un planificateur de PRM, on définit deux fonctions pour que les algorithmes soient clairs [9] :

- $\Delta : \mathcal{C}_{free} \times \mathcal{C}_{free} \rightarrow \{chemin, NIL\}$, Δ est un planificateur local, restituant le chemin qui n'est pas en collision entre deux configurations appartenant à $\mathcal{C}_{free}$ si tel chemin existe ; ou il restitue *NIL* s'il ne trouve pas de chemin.
- $\mathcal{D} : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}^+ \cup \{0\}$,$\mathcal{D}$ est *la fonction Distance* qui calcule la distance entre deux configurations dans l'espace $\mathcal{C}$; souvent elle est une distance *métrique.*

Les méthodes basées sur la carte de route probabiliste comme c'est présenté dans [2] sont principalement constituées de deux phases : la phase d'apprentissage et la phase de traitement des requêtes.

3.1.1 La phase d'apprentissage

Dans la phase d'apprentissage, la carte de route probabiliste est construite et sauvegardée sous forme d'un graphe $G(V, E)$ suivant deux étapes : *la construction* et *l'expansion.* Et à son tour l'étape de construction se constitue de deux étapes : *l'échantillonnage* et *la connexion entre les échantillons.*

Dans l'étape d'échantillonnage, chacune des méthodes utilisées génère de façon aléatoire des configurations dans l'espace $\mathcal{C}_{free}$ et les sauvegarde dans V comme des noeuds, et là les techniques diffèrent en termes d'admission de la configuration qui n'est pas en collision.

Dans l'étape de connexion entre les échantillons, un planificateur local est utilisé pour réaliser les connexions entre chaque configuration $q \in V$ avec un ensemble N_q de ses adjacents. Ces adjacents sont sélectionnés selon une distance *dist* (nous allons présenter dans ce chapitre quelques formules pour la fonction $\mathcal{D}$ qui calcule la distance *dist*). La taille de l'ensemble N_q est limitée par un nombre k choisi. Si le planificateur réussit à trouver un chemin entre q et $q\prime_i$ alors l'arrête $(q, q\prime_i)$ est ajoutée à la matrice des adjacents E (on peut aussi utiliser une liste au lieu d'une matrice). On appelle le chemin trouvé par le planificateur local entre q et $q\prime_i$ un *chemin local.*

L'étape d'expansion consiste à améliorer la connectivité du graphe G, en utilisant une

fonction heuristique qui évalue les noeuds de G, pour voir quels sont les configurations qui se situent dans les régions difficiles, puis, l'expansion de la carte s'effectue par la génération de nouvelles configurations au voisinage de ces configurations.

L'algorithme d'apprentissage présenté dans [2], fonctionne de telle sorte qu'à chaque itération, il génère une configuration libre et il essaie de la connecter au graphe G, jusqu'à n. Mais dans [9] l'algorithme génère d'abord n configurations libres, puis il essaie de les connecter entre elles. Nous avons utilisé dans notre mémoire l'algorithme de [9] (Algorithme 3).

Dans l'algorithme 3 on suppose que le planificateur local est symétrique et déterministe.

Algorithm 3 Construction de la carte

Entrées:

n : Nombre des configurations à générer

k : Nombre des proximités à examiner pour chaque configuration

Sorties :

$G(V, E)$

1: $E \leftarrow \{\}$;
2: $V \leftarrow$ L'ensemble de n configurations générées par une méthode d'échantillonnage ;
3: **Pour toute** $q \in V$ **Faire**
4: $N_q \leftarrow k$ proximité de q Selon *dist* ;
5: **Pour** chaque $q\prime \in N_q$ **Faire**
6: **Si** $(q, q\prime) \notin E$ et $\Delta(q, q\prime) \neq NIL$ **Alors**
7: $E \leftarrow E \cup \{(q, q\prime)\}$;
8: **Fin Si**
9: **Fin Pour**
10: **Fin Pour**

Dans l'algorithme 3, les variables n (ligne 2), k et *dist* (ligne 4), jouent un rôle important dans la puissance de connexité de la carte, tel que si on donne à ces variables des valeurs importantes, on obtient une carte de route très puissante, mais ça prend beaucoup de temps pour la construire, alors, il faut choisir ces variables de tel sorte que la carte doit être construite dans un temps raisonnable. Le choix pratique qui est souvent utilisé, est de choisir le nombre des configurations n plus grand possible et le nombre des adjacents k et la distance *dist* petits.

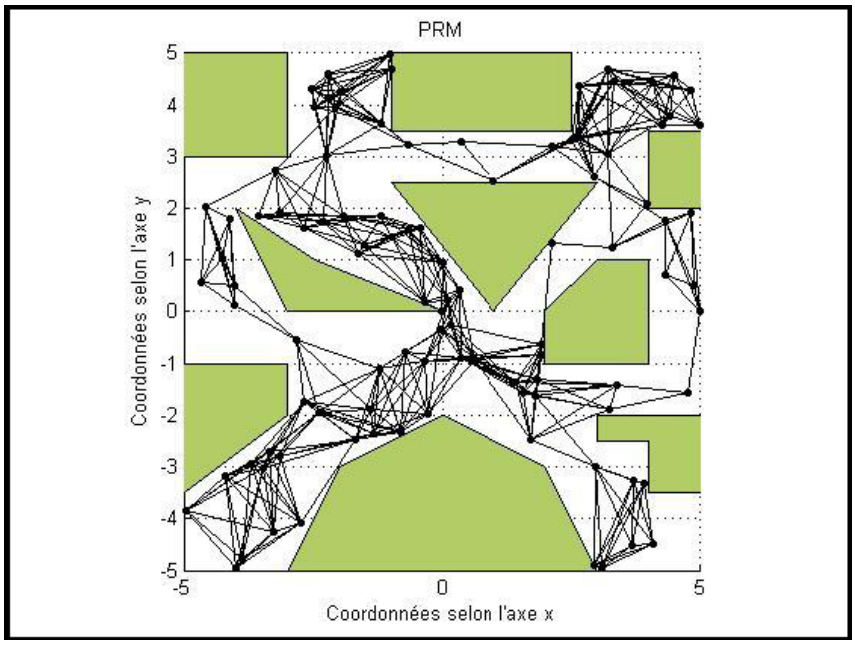

FIGURE 3.1 – Une carte de route probabiliste pour un robot ponctuel dans un environnement en 2D.

3.1.2 La phase de traitement des requêtes

Dans la phase de traitement des requêtes (Algorithme 4), on utilise le planificateur local précédent Δ pour connecter les deux configurations initiales et finales (q_{init}, q_{goal}) au graphe G ; ensuite, on utilise un algorithme qui recherche le plus court chemin dans un graphe (comme : Dijkstra,A^*,...) pour trouver la séquence des configurations $q_c = \{q_{init}, q_{c_1}, \ldots, q_{c_m}, q_{goal}\}$ qui forme le plus court chemin dans G entre q_{init} et q_{goal} s'il existe. Puis, une fonction de lissage est utilisée pour transformer la séquence des configurations, qui forme le chemin, à une trajectoire $q(t)$ de classe C^2 pour le robot.

Algorithm 4 La phase de traitement des requêtes

Entrées:

q_{init} : La configuration initiale

q_{goal} : La configuration finale

k : Nombre des adjacents à examiner pour chaque configuration

$G(V, E)$: La carte construite par l'algorithme 3

Sorties :

Le chemin entre q_{init} et q_{goal} ; ou l'échec ;

1: $V \leftarrow V \cup \{q_{init}\} \cup \{q_{goal}\}$;

2: **Pour** $q \leftarrow \{q_{init}, q_{goal}\}$ **Faire**
3: $N_q \leftarrow k$ adjacents de q dans V selon *dist* ;
4: $q\prime \leftarrow$ l'adjacent le plus proche de q dans N_q ;
5: **Tant que** N_q n'est pas vide **Faire**
6: **Si** $\Delta(q, q\prime) \neq NIL$ **Alors**
7: $E \leftarrow E \cup (q, q\prime)$;
8: $N_q \leftarrow N_q - \{q\prime\}$;
9: **Si non**
10: $N_q \leftarrow N_q - \{q\prime\}$;
11: **Fin Si**
12: $q\prime \leftarrow$ L'adjacent le plus proche dans N_q ;
13: **Fin Tant que**
14: **Fin Pour**

15: Chemin $\leftarrow$ Le plus courte chemin (q_{init}, q_{goal}, G)
16: **Si** Chemin n'est pas vide **Alors**
17: **Returner** Chemin ;
18: **Si non**
19: **Returner** Echec ;
20: **Fin Si**

Si la phase de traitement des requêtes échoue fréquemment, alors c'est une indication que la carte de route ne peut pas capturer la connectivité de $\mathcal{C}_{free}$ de manière adéquate. Donc, plus de temps devrait être consacré à la phase d'apprentissage. Mais cela ne signifie pas qu'une nouvelle carte doit être construite dès le début. Lorsque l'apprentissage est incrémental, on peut étendre la carte courante en recommençant l'algorithme de construction (Algorithme 3 sans la ligne 1 et 2) [2].

3.1.3 Post-traitement

L'étape de post-traitement est appliquée sur le chemin obtenu par l'algorithme 4 pour améliorer sa qualité, et cette étape s'effectue par le test avec le planificateur local Δ s'il existe un chemin entre chaque deux configurations dans la séquence du chemin qui ne sont pas adjacents. L'approche qu'on a utilisée pour cette étape est l'approche *Greedy* [9], qui consiste à fixer au début q_{goal} comme une configuration but, et d'essayer de la connecter avec les autres configurations en commençant par la première, ensuite, la confi-

guration avant l'ancienne configuration but devient la configuration but, et on refait la même démarche ; quand une connexion entre deux configurations réussit on élimine les configurations qui se trouvent entre elles.

Algorithm 5 Le post-traitement qui s'effectue sur le chemin obtenu par l'algorithme 4

Entrées: Chemin

Sorties : Chemin plus optimal que l'ancien

1: $i \leftarrow 1$;
2: $n \leftarrow$ La taille du Chemin ;
3: $j \leftarrow n$;
4: **Tant que** $i < n - 1$ **Faire**
5: **Tant que** $j > i + 1$ **Faire**
6: **Si** $\Delta(Chemin(i), Chemin(j)) \neq NIL$ **Alors**
7: $Chemin(i + 1 : j - 1) \leftarrow [\,]$;
8: **Break** ;
9: **Fin Si**
10: $j \leftarrow j - 1$;
11: **Fin Tant que**
12: $j \leftarrow n$;
13: $i \leftarrow i + 1$;
14: **Fin Tant que**

3.2 L'échantillonnage

Les planificateurs probabilistes réussissent à résoudre le problème de planification du chemin pour les robots ayant plusieurs degrés de liberté, parce que ces planificateurs échantillonnent aléatoirement l'espace des configurations. Alors le composant crucial de ces planificateurs est la fonction qui génère de façon incrémentale des configurations aléatoires dans l'espace $\mathcal{C}$. Un générateur de nombres *pseudo-aléatoire* standard peut être utilisé pour générer des échantillons, et pour une meilleure *uniformité* en termes de dispersion et de discrépance, une séquence déterministe de nombres quasi-aléatoires peut aussi être utilisée, tel que : la séquences de *Halton*, la séquences de *Van der Corput*, et la séquences de *Hammersley* [9].

La plupart des stratégies d'échantillonnage nécessitent un système de détection de collision pour savoir si une configuration est en collision ou non. La fonction Γ (Equation

3.2) peut être considérée comme un système de détection de collision qui retourne 1 si une configuration est en collision et 0 si elle ne l'est pas.

$$\Gamma(q) = \begin{cases} 1 & \text{si } q \notin \mathcal{C}_{free} \\ 0 & \text{si } q \in \mathcal{C}_{free} \end{cases} \tag{3.1}$$

3.2.1 Echantillonnage avec une distribution uniforme

La loi uniforme est une loi de probabilité très simple puisque c'est celle du pur hasard.

Definition 3.2.1. [23] *On dit qu'une variable aléatoire x est générée par une loi de probabilité uniforme sur l'intervalle $[a, b]$, si la densité de probabilité $f(x)$ est toujours égale sur cet intervalle et nulle en dehors.*

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{si } a < x < b \\ 0 & \text{en dehors} \end{cases} \tag{3.2}$$

L'échantillonnage avec une distribution uniforme est utilisé dans les premières recherches sur les cartes de route probabilistes [2] parce qu'elle est la méthode la plus facile à implémenter, en plus elle résout beaucoup de problèmes de planification, donnant de bons résultats [9].

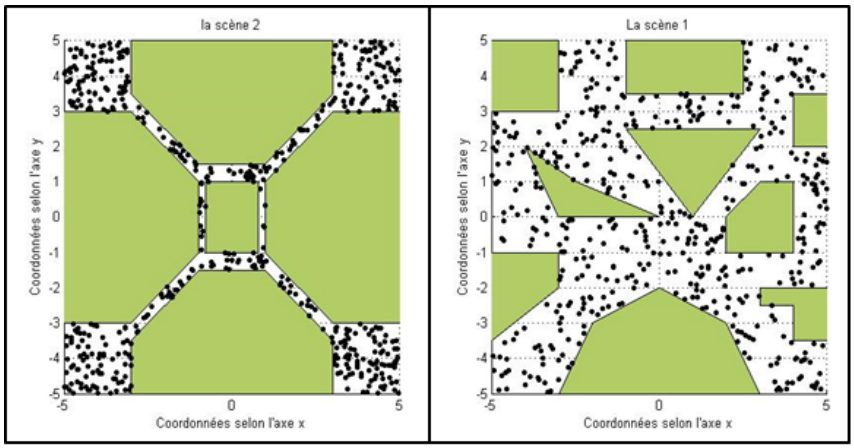

FIGURE 3.2 – 500 configurations d'un robot ponctuel générées par la une distribution uniforme, avec deux scènes différentes.

Supposant qu'on veut échantillonner uniformément l'espace cartésien $\mathcal{C} = \mathbb{R}^3$, alors on utilise simplement la loi uniforme précédente pour générer x, y et z, mais si on veut échantillonner uniformément l'espace $SO(3)$, certains soins doivent être pris [24].

Dans notre cas, l'environnement est en $2D$, alors le robot a deux coordonnées pour la position et un angle pour l'orientation, on parle alors de l'espace $SE(2)$. Pour échantillonner uniformément cet espace on génère simplement x et y par une distribution uniforme, ensuite on génère l'angle d'orientation $\theta \in [-\pi, \pi]$ par la même distribution.

Il existe des cas où l'échantillonnage uniforme a un mauvais rendement. Souvent, cela est le résultat de ce qu'on appelle le problème de *passage étroit*. Si un passage étroit existe dans $\mathcal{C}_{free}$ et il est absolument nécessaire de passer par ce passage pour résoudre une requête ; une méthode d'échantillonnage doit échantillonner densément dans le passage étroit afin de répondre à la requête de planification. Un certain nombre de stratégies d'échantillonnage ont été proposées pour résoudre ce problème.

Nous présentons deux stratégies qui traitent ce problème : l'échantillonnage gaussien (gaussian sampling) [25], l'échantillonnage par pont (bridge test sampling) [26].

3.2.2 Echantillonnage gaussien

La méthode d'échantillonnage gaussien [25] consiste à générer une configuration aléatoire c_1 par une distribution uniforme. Ensuite, une distance d est choisie selon une distribution normale pour générer une autre configuration aléatoire c_2 qui se trouve à une distance d de la configuration c_1. Les deux configurations sont rejetées si les deux sont en collision ou si les deux ne sont pas en collision. Si l'une des deux configurations est en collision et l'autre n'est pas en collision alors on ajoute la configuration qui n'est pas en collision à la carte.

Definition 3.2.2. [23] *On dit qu'une variable aléatoire x est générée par une loi de probabilité normale si la densité de probabilité $f(x)$ est de forme :*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, x \in \mathbb{R} \tag{3.3}$$

Tel que μ est la moyenne et σ est l'écart type.

L'écart type σ est une variable très importante, car c'est le paramètre qui permet de

Algorithm 6 [25] Echantillonnage Gaussien

Entrées:

n : Nombre des configurations à générer

Sorties :

V : Liste de n configurations aléatoires

Tant que $|V| < n$ **Faire**
  $c_1 \leftarrow$ Configuration aléatoire générée par une distribution uniforme ;
  $d \leftarrow$ Distance générée par une distribution normale ;
  $c_2 \leftarrow$ Configuration aléatoire se trouvant à une distance d de c_1 ;
  Si $c_1 \in c_{free}$ et $c_2 \notin c_{free}$ **Alors**
    $V \leftarrow V \cup \{c_1\}$;
  Si non
    Si $c_2 \in c_{free}$ et $c_1 \notin c_{free}$ **Alors**
      $V \leftarrow V \cup \{c_2\}$;
    Fin Si
  Fin Si
Fin Tant que

rapprocher ou d'éloigner les configurations des obstacles : si nous diminuons sigma, les configurations se rapprochent des obstacles, et si nous augmentons sigma les configurations s'en éloignent.

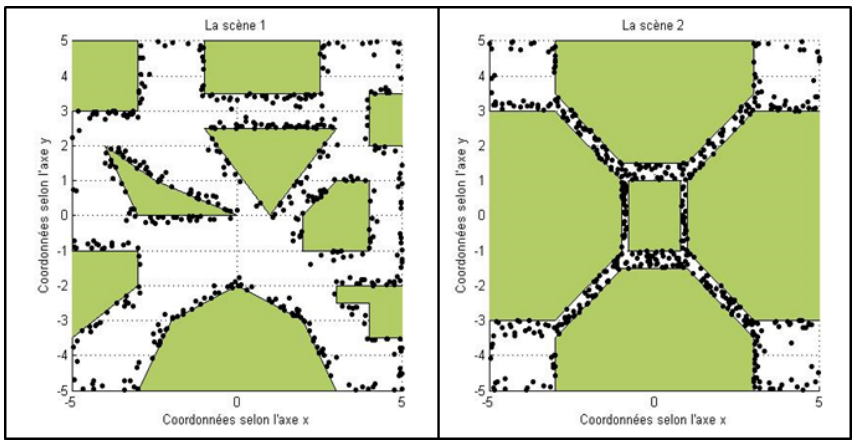

FIGURE 3.3 – 500 configurations d'un robot ponctuel générées par la méthode d'échantillonnage gaussien, avec deux scènes différentes.

3.2.3 Echantillonnage par pont

L'échantillonnage par pont [26] utilise un pont pour échantillonner à l'intérieur des passages étroits. Le principe de cette méthode est de générer une configuration c_1 par une distribution uniforme, ensuite si cette configuration est en collision, alors une autre configuration c_2 est générée au voisinage de c_1 par une loi de probabilité de densité λ_x (souvent λ_x est gaussien de centre c_1 et d'un écart type σ petit). Si cette deuxième configuration est aussi en collision, alors on accepte le centre c du segment $\overline{c_1c_2}$ s'il n'est pas en collision.

La cause du choix σ petit si λ_x est gaussien, est de construire des ponts courts pour échantillonner dans les passages étroits. Le choix de σ dépend de la taille des passages étroits qui existent dans la scène choisie. Ainsi cette méthode risque d'entrer dans une boucle infinie lorsqu'elle essaie d'échantillonner dans les passages étroits par un pont très court (Figure3.4).

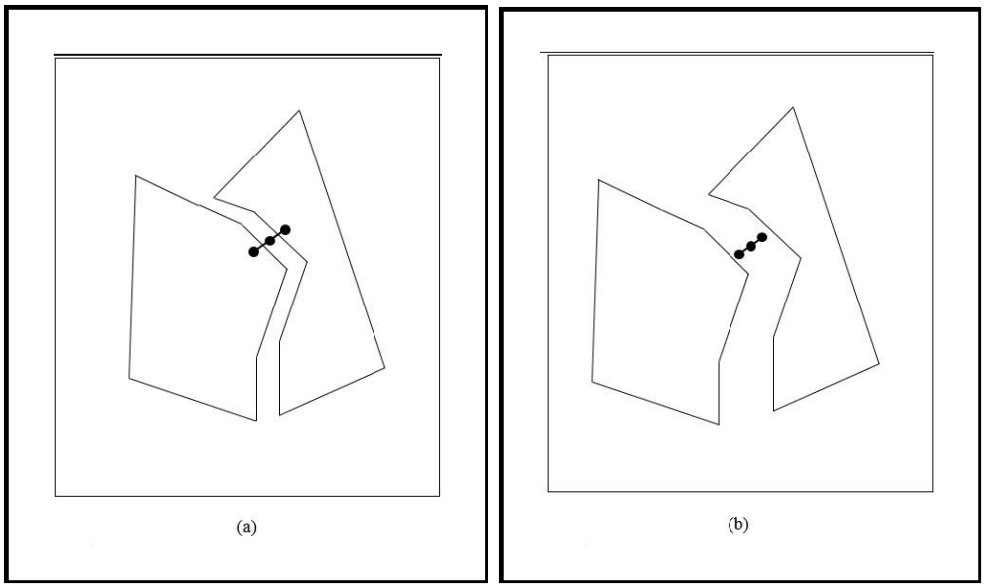

FIGURE 3.4 – (a) Le pont est court mais il a réussi à échantillonner dans le passage étroit. (b) Le pont n'a pas réussi à échantillonner dans le passage étroit car il est plus court que le passage.

Algorithm 7 [26] Echantillonnage par pont

Entrées:

n : Nombre des configurations à générer

Sorties :

V : Liste de n configurations aléatoires

Tant que $|V| < n$ **Faire**
  $c_1 \leftarrow$ Configuration aléatoire générée par une distribution uniforme ;
  Si $c_1 \in c_{obs}$ **Alors**
    $c_2 \leftarrow$ Configuration aléatoire générée au voisinage de c_1 par une loi de probabilité de densité λ_x ;
    Si $c_2 \in c_{obs}$ **Alors**
      $c \leftarrow$ le centre du segment $\overline{c_1 c_2}$;
      Si $c \in c_{free}$ **Alors**
        $V \leftarrow V \cup \{c\}$;
      Fin Si
    Fin Si
  Fin Si
Fin Tant que

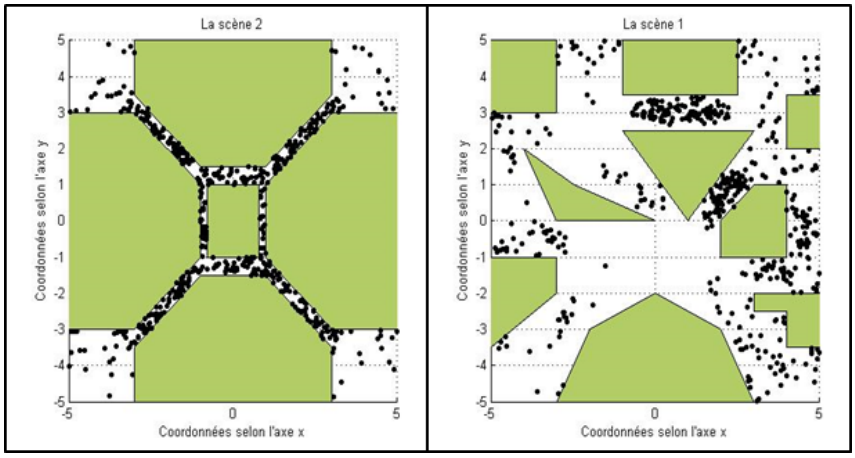

FIGURE 3.5 – 500 configurations d'un robot ponctuel générées par la méthode d'échantillonnage par pont, avec deux scènes différentes.

3.3 Les planificateurs locaux

Les planificateurs locaux sont utilisés pour établir des connexions entre les noeuds lors de la construction de la carte de route, et aussi entre les deux configurations q_{init}, q_{goal} et

la carte de route lors du traitement des requêtes. On peut distinguer deux types de planificateurs locaux : les planificateurs locaux *simples* et les planificateurs locaux *puissants*.

Si un planificateur local puissant est utilisé, il arrive souvent à trouver un chemin lorsqu'il existe. Par conséquent, peu de noeuds peuvent être nécessaires pour construire une carte de route suffisante à répondre de manière fiable à des requêtes de planification. Ce planificateur est un peu lourd, mais lorsque la carte n'est pas large, l'algorithme de planification n'a pas besoin de nombreux appels à ce planificateur. D'autre part, le planificateur simple est très rapide mais avec moins de succès, alors il faudra plus de configurations à inclure à la carte de route, et par conséquent, le planificateur local est appelé plusieurs fois pour réaliser les connexions entre les noeuds [9].

On a supposé dans l'algorithme 3 que le planificateur local est symétrique et déterministe, il est également possible d'utiliser un planificateur local asymétrique et non déterministe. Dans de nombreux cas, la connexion d'une configuration q à une configuration $q\prime$ ne signifie pas nécessairement que le contraire peut être fait. Si le planificateur local ramène le robot de q à $q\prime$, et le robot peut également exécuter le chemin en sens inverse pour aller de $q\prime$ à q, alors la carte de route est un graphe non orienté, c'est-à-dire l'ajout de l'arrête (q,$q\prime$) est le même que l'ajout de l'arrête ($q\prime$, q). Si les chemins locaux ne peuvent pas être exécutés en inverse, alors il faut utiliser un graphe orienté pour modéliser la carte de route [9].

On sait qu'un planificateur local déterministe retourne toujours le même chemin entre deux configurations, donc la carte de route n'a pas besoin de mémoriser ce chemin local, tel que le chemin peut être recalculé pour répondre à une requête de planification. D'autre part, si un planificateur local non déterministe est utilisé, la carte de route devra associer à chaque arrête le chemin local calculé par Δ. En général, l'utilisation des planificateurs locaux non déterministes augmente les exigences de stockage de la carte de route [9].

Definition 3.3.1. [9] *Soit Δ un planificateur local, et soit $q, q\prime \in \mathcal{C}_{free}$ et $\Delta(q, q\prime) \neq NIL$: On dit que Δ est symétrique si le robot peut se déplacer de q à $q\prime$, en exécutant le chemin calculé par Δ, et de $q\prime$ à q en exécutant le même chemin en inverse.*

Definition 3.3.2. [9] *Soit Δ un planificateur local, et soit $q, q\prime \in \mathcal{C}_{free}$:*
On dit que Δ est déterministe s'il produit toujours le même chemin entre q et $q\prime$.

Dans les deux sections suivantes, nous présenterons deux planificateurs simples et

déterministes que nous avons implémenté dans notre mémoire : planificateur de ligne droite (Straight-line), et le planificateur de rotation-à-S (Rotate-at-s).

3.3.1 Le planificateur local de ligne droite

Ce planificateur est le plus populaire parce que son principe est très facile : il relie entre chaque deux configurations c_1, c_2 par un segment de droite dans l'espace $\mathcal{C}$ et le teste s'il est en collision par la discrétisation de ce segment en un nombre des configurations $(c_1, q_1, \ldots, c_n, q_2)$, tel que la distance entre chaque deux configurations successives est inférieure à un nombre positif petit appelé *Pas*, et ce nombre choisi expérimentalement de telle sorte qu'il garantit l'absence d'un obstacle. Puis, il teste la collision pour chaque configuration de la séquence $(c_1, q_1, \ldots, c_n, q_2)$ qui forme le segment. Deux techniques sont souvent utilisées pour choisir la configuration à tester : l'incrémentation et la subdivision. Il faut faire attention à l'interpolation, tel que les composants de translation et les composants de rotation sont interpolés séparément [27][9].

Pour la technique d'incrémentation, le planificateur teste la séquence $(c_1, q_1, \ldots, c_n, q_2)$ de façon incrémentale, c'est-à-dire qu'à chaque fois il ajoute à la configuration testée le nombre positif *Pas* et il teste la collision pour cette nouvelle configuration jusqu'à ce qu'un obstacle soit détecté ou qu'il arrive à la fin du segment.

Pour la technique de la subdivision, le planificateur au début teste la collision pour le centre du segment $\overline{c_1 c_2}$, puis à chaque fois il teste les centres des sous-segments obtenus, jusqu'à ce qu'un obstacle soit détecté ou la longueur du sous-segment soit inférieure au nombre positif *Pas*.

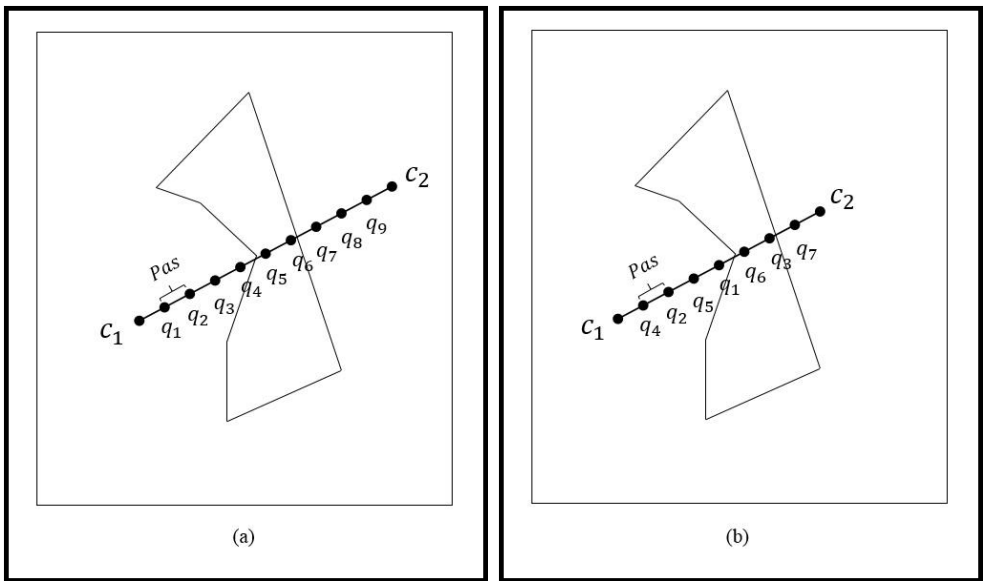

FIGURE 3.6 – Test de collision pour le planificateur local ligne droite : (a) technique d'incrémentation. (b) technique de la subdivision.

Algorithm 8 [27] Le test de collision du segment $\overline{c_1 c_2}$ s'effectue par le planificateur ligne droite avec la technique d'incrémentation.

Entrées: c_1, c_2

Sorties : OUI (si aucun obstacle n'est détecté) / NON (si un obstacle est détecté)

Fonction LIGNE DROITE(c_1, c_2)
  $n \leftarrow$ Le nombre des configurations intermédiaires ;
  $Pas \leftarrow$ Les valeurs d'incrémentation de chaque élément de la configuration ;
  $q \leftarrow c_1 + Pas$;
  Pour $i = 1$ **jusqu'à** n **Faire**
    $q \leftarrow q + Pas$;
    Si q en collision **Alors**
      Returner NON ;
    Fin Si
  Fin Pour
  Returner OUI ;
Fin Fonction

3.3.2 Le planificateur local de rotation-à-S

Avec le planificateur ligne droite, le robot réalise les mouvements de translation et de rotation en même temps durant le déplacement de c_1 vers c_2. Mais avec le planificateur

de rotation-à-s le robot fait une translation vers une configuration intermédiaire $c\prime$; puis le robot se tourne de sorte à prendre l'orientation de c_2, cette deuxième configuration intermédiaire est désignée par $c\prime\prime$; ensuite il effectue une deuxième translation vers c_2. Le paramètre $s \in [0, 1]$ permet de déterminer la position de $c\prime$ sur le segment $\overline{c_1 c_2}$ par la multiplication de s fois la distance de translation entre c_1 et c_2. Généralement, le paramètre s est choisi tel que $c\prime$ se trouve au début ou au milieu ou bien à la fin du segment ($s = 0; 0.5; 1$). Le planificateur local de ligne droite peut être utilisé pour planifier au niveau de chaque paire de configuration $(c_1, c\prime)$, $(c\prime, c\prime\prime)$ et $(c\prime\prime, c_2)$ [27].

Algorithm 9 [27] Le test de collision qui s'effectue par le planificateur rotation-à-s

Entrées: c_1, c_2, s

Sorties : OUI (si aucun obstacle n'est détecté) / NON (si un obstacle est détecté)

Fonction ROTATION-À-S(c_1, c_2, s)
  $(x\prime, y\prime, z\prime) \leftarrow s(x_2 - x_1, y_2 - y_1, z_2 - z_1)$;
  $c\prime \leftarrow (x\prime, y\prime, z\prime, \alpha_1, \beta_1, \gamma_1)$;
  $c\prime\prime \leftarrow (x\prime, y\prime, z\prime, \alpha_2, \beta_2, \gamma_2)$;
  $p_1 \leftarrow$Ligne droite$(c_1, c\prime)$;
  $p_2 \leftarrow$Ligne droite$(c\prime, c\prime\prime)$;
  $p_3 \leftarrow$Ligne droite$(c\prime\prime, c_2)$;
  Si p_1, p_2, p_3 sont tous OUI **Alors**
    Returner OUI;
  Si non
    Returner NON ;
  Fin Si
Fin Fonction

3.4 La fonction Distance D

La distance calculée par la fonction $\mathcal{D}$ est utilisée dans les PRMs pour déterminer quels adjacents d'une configuration on tente de connecter à l'aide d'un planificateur local. Ainsi, elle joue un rôle crucial dans l'efficacité et la réussite de la PRM. Une bonne distance permet de limiter le nombre d'appels au planificateur local par la classification selon la proximité des adjacents, et elle permet de produire une carte de route bien connectée. La fonction $\mathcal{D}$ doit calculer la distance très rapidement, parce que le calcul de la distance est l'une des opérations les très fréquemment sollicitées dans une PRM. Le choix d'une bonne

distance est entravé par le fait que l'espace $\mathcal{C}$ ne soit pas un espace euclidien, et donc nos notions intuitives de proximité ne sont pas nécessairement significatives [2].

Une possibilité consiste à définir la distance entre deux configurations $\mathcal{D}(q\prime, q\prime\prime)$ par certaines mesures des zones balayées par le robot dans d'espace $\mathcal{W}$, comme la surface ou le volume, lorsqu'il se déplace en l'absence d'obstacles sur le trajet $\Delta(q\prime, q\prime\prime)$. Par conséquent, la minimisation du volume balayé permet de réduire la chance de collision [9].

Le calcul exact des surfaces ou des volumes balayés dans l'espace $\mathcal{W}$ est notoirement difficile, ce qui explique pourquoi les mesures heuristiques tentent généralement d'estimer la métrique de la surface ou du volume balayé. Pour cela, une approche très commune est de considérer l'espace de configurations $\mathcal{C}$ comme un espace cartésien et d'utiliser la distance euclidienne [27].

Dans le cas où le robot est un corps rigide dans l'espace $3D$, une configuration de ce corps a la forme $(x, y, z, \alpha, \beta, \gamma) \in \mathbb{R}^6$, et une distance métrique dans $\mathcal{C}$ entre deux configurations $q\prime$ et $q\prime\prime$ de ce robot peut être calculée par les trois formules prés de [27] présentés dans le tableaux suivant :

TABLE 3.1 – Trois formules pour calculer la distance métrique dans $\mathcal{C}$.

La distance euclidienne	$\mathcal{D}(q\prime, q\prime\prime) = \left(\sum_{k=x,y,z} P(k)^2 + \sum_{k=\alpha,\beta,\gamma} Q(k)^2\right)^{\frac{1}{2}}$
La distance de Minkowski	$\mathcal{D}(q\prime, q\prime\prime) = \left(\sum_{k=x,y,z} P(k)^r + \sum_{k=\alpha,\beta,\gamma} Q(k)^r\right)^{\frac{1}{r}}$
La distance de Manhattan	$\mathcal{D}(q\prime, q\prime\prime) = \sum_{k=x,y,z} P(k) + \sum_{k=\alpha,\beta,\gamma} Q(k)$

Tel que $Q(k) = n \mid q\prime(k) - q\prime\prime(k) \mid \text{et} P(k) = \mid q\prime(k) - q\prime\prime(k) \mid$. Et n est le facteur de normalisation pour garder les trois angles d'orientations α, β, γ entre 0 et 1.

La distance de Minkowski est une distance euclidienne généralisée par le remplacement de la puissance 2 par le paramètre r, et la distance de Manhattan c'est la distance de Minkowski avec r tendant vers l'infini.

3.5 Conclusion

Nous avons présenté dans ce chapitre les détailles de fonctionnement des planificateurs de carte de routes probabiliste avec trois méthodes d'échantillonnage et deux planificateur locaux, ensuite nous avons présenté trois formules pour calculer la distance entre deux configurations dans l'espace des configurations.

Chapitre 4

Réalisation et simulation

Un robot manipulateur est constitué d'un système mécanique articulé et d'un ensemble d'organes associés. La planification de trajectoire pour un robot manipulateur se fait par la planification de trajectoire pour chaque organe.

Dans ce chapitre nous allons présenter notre réalisation d'un planificateur de chemin basé sur la PRM pour un robot modélisé par un polygone dans un environnement en 2D, et ceci en implémentant les méthodes d'échantillonnage et les planificateurs locaux présentés dans le chapitre 3, ainsi que les algorithmes de recherche du plus court chemin présentés dans le chapitre 2, ensuite nous allons faire quelques comparaisons entre les performances de ces méthodes.

Les méthodes d'échantillonnage et les planificateurs locaux que nous avons utilisé sont choisis d'après les résultats de [27][10].

4.1 Plan d'exécution d'un planificateur de PRM

D'après [2], la phase d'apprentissage et la phase de traitement des requêtes peuvent être exécutées en entrelacement, en revenant à l'étape d'expansion à chaque fois que la phase de traitement des requêtes échoue. Mais dans notre réalisation, on considère que les deux phases sont exécutées séquentiellement, en ignorant l'étape de l'expansion. Le schéma de la figure 4.1 résume le plan d'exécution d'un planificateur de chemin basé sur la PRM.

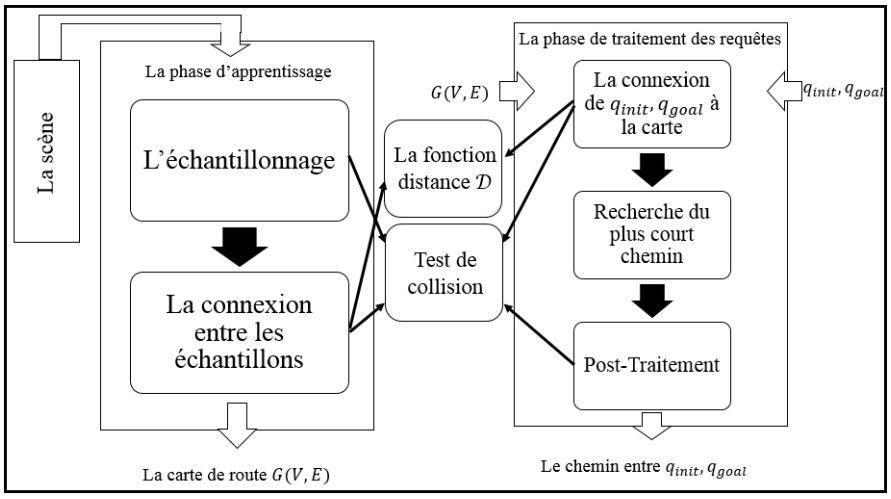

FIGURE 4.1 – Plan d'exécution d'un planificateur de chemin basé sur la PRM.

4.2 L'échantillonnage

4.2.1 Génération des nombres aléatoires

Pour les générateurs de nombres aléatoires, on a utilisé les fonctions de MATLAB, telles que la fonction*rand* qui génère des nombres aléatoires entre 0 et 1 selon une distribution uniforme, et la fonction *randn* qui génère des nombres aléatoires selon une distribution normale de centre 0.

4.2.2 Génération de c_2 avec l'échantillonnage Gaussien

Le problème majeur dans cette méthode est la génération aléatoire de la deuxième configuration c_2 qui se trouve à une distance d de la première configuration c_1. L'idée que nous avons utilisé dans le cas où le robot est ponctuel dans un environnement en $2D$ ($\mathcal{C}$ est un espace cartésien de dimension 2), est de générer aléatoirement la coordonnée x_2 de la deuxième configuration dans l'intervalle $[-d, d]$, puis on calcule la coordonné y_2 en utilisant le théorème de Pythagore (si la distance euclidienne est choisie pour la fonction distance $\mathcal{D}$) : c_2 se situe sur le cercle de rayon d et de centre c_1.

Algorithm 10 Pseudo-code pour générer c_2 dans l'algorithme 6

1: $x_1 := s * rand() * X_{max}$;

2: $y_1 := s * rand() * Y_{max}$;

3: $d = randn()$;

4: $x_2 := s * rand() * d + x_1$;

5: $y_2 := s * \sqrt{d^2 - x_2^2} + y_1$;

X_{max} ,Y_{max} : Sont les limites de la scène

s : Est un signe aléatoire

Dans le cas où le robot est un polygone avec une scène en $2D$, l'espace $\mathcal{C}$ devient de dimension 3 (deux coordonnées pour la position x, y et une pour l'orientation θ), la génération de la deuxième configuration sera un peu difficile. L'idée que nous avons proposée consiste à générer deux éléments de c_2 par une distribution normale, tel que le centre de la distribution qui génère l'élément choisi de c_2 est l'élément qui correspond à celui-ci dans c_1 ; puis on calcule le troisième élément de c_2 en utilisant la formule de la fonction distance $\mathcal{D}$. Cette technique peut être généralisée pour les espaces de configurations à plusieurs dimensions.

4.3 La subdivision et la direction

Quand le robot est ponctuel, le planificateur Rotation-à-s n'a aucun sens, parce que l'orientation du robot n'a aucun effet sur le test de collision ; et quand le robot et un corps rigide dans une scène en $2D$, le problème de la direction apparaît. Notre choix est pris de [24] qui consiste à choisir la direction au sens du petit angle entre les deux configurations c_1 ,c_2.

L'algorithme 8 utilise la technique d'incrémentation pour tester la collision du segment $\overline{c_1 c_2}$. Ici nous présentons l'algorithme qu'on a implémenté tel qu'il utilise la technique de la subdivision, en considérant que le robot est un polygone dans un environnement en $2D$ (les configurations sous forme (x, y, θ)).

Algorithm 11 Le test de collision du segment $\overline{c_1c_2}$ s'effectue par le planificateur de ligne droite avec la technique de la subdivision.

Entrées: c_1, c_2

Sorties : OUI (si aucun obstacle n'est détecté) / NO (si un obstacle est détecté)

Fonction LIGNE DROITE(c_1, c_2)
  $\psi_1 \leftarrow |\theta_1 - \theta_2|$;
  $\psi_2 \leftarrow 2\pi - \psi_1$;
  Si $\psi_1 > \psi_2$ et $\theta_1 < \theta_2$ **Alors**
    $\theta_1 \leftarrow \theta_1 + 2\pi$;
  Si non
    Si $\psi_1 > \psi_2$ et $\theta_1 > \theta_2$ **Alors**
      $\theta_2 \leftarrow \theta_2 + 2\pi$;
    Fin Si
  Fin Si
  $Segm \leftarrow \{c_1, c_2\}$;
  Tant que $\mathcal{D}(Segm(1), Segm(2)) > Pas$ **Faire**
    $Segm1 \leftarrow \{\}$;
    $N \leftarrow$ La taille de Segm
    Pour $i \leftarrow 1$ **jusqu'à** $N - 1$ **Faire**
      Centres(i)$\leftarrow$ (Segm(i),Segm(i+1))/2 ;
      Segm1 $\leftarrow \{Segm1, Segm(i), Centres(i)\}$;
    Fin Pour
    $S \leftarrow$ La taille de Centres ;
    Pour $i \leftarrow 1$ **jusqu'à** S **Faire**
      Si $Centres(i)$ est en collision **Alors**
        Returner NO ;
      Fin Si
    Fin Pour
    $Segm \leftarrow \{Segm1, c_2\}$;
  Fin Tant que
  Returner OUI ;
Fin Fonction

4.4 Test de collision

Pour le test de collision, on a utilisé les fonctions MATLAB (*inpolygon, polyxpoly, polybool*), tel que :

inpolygon : Teste la collision d'un point avec un polygone (elle utilise l'algorithme de [19]).

polyxpoly : Teste la collision d'un segment de droite avec les arêtes ou les sommets d'un polygone.

polybool : Fait plusieurs opérations sur les régions polygonales. Peut-être utilisée pour le test de collision d'un polygone avec un autre polygone.

4.5 L'absence du chemin

L'algorithme de Dijksrta présenté dans le chapitre 2 ne peut pas détecter l'absence du chemin si le graphe est non connecté. Ici on présente un algorithme modifié de Dijkstra qui peut détecter l'absence du chemin. Notant que l'algorithme de A^* (Algorithme 2) peut détecter l'absence du chemin.

Algorithm 12 Pseudo-code de l'algorithme de Dijkstra

Entrées: $G(V, E)$,S_init, S_désiré

Sorties : Chemin

```
1: Liste_ouverte := {S_init};
2: Liste_fermée := {S_init};
3: g := {0};
4: Liste_provenance := { }; Noeuds := { };
5: Tant que la Liste_ouverte n'est pas vide Faire
6:     Courant := le sommet dans Liste_ouverte qui a le g minimum;
7:     Pour chaque adjacent de courant Faire
8:         Si l'adjacent est dans la Liste_fermée Alors
9:             continue;
10:        Fin Si
11:        Dist := E(Courant, adjacent) + g[Courant];
12:        Si l'adjacent est dans Liste_ouverte Alors
13:            Si Dist < g[adjacent] Alors
```

```
14:                 g[adjacent]:= Dist ;
15:                 Liste_provenance [adjacent] := Courant ;
16:                 continue ;
17:             Si non
18:                 continue ;
19:             Fin Si
20:         Fin Si
21:         Ajouter l'adjacent à la Liste_ouverte ;
22:         Ajouter Dist à g ;
23:         Ajouter Courant à la Liste_provenance et ajouter l'adjacent à Noeuds ;
24:     Fin Pour
25:     Liste_ouverte[Courant] := [ ] ;
26:     Ajouter Courant à la Liste_fermée ;
27:     g[Courant] := [ ] ;
28: Fin Tant que
29: Returner reconstruireChemin(Liste_provenance, Noeuds) ;

30: Fonction ECONSTRUIRECHEMIN(Liste_provenance, Noeuds)
31:     Chemin := {S_désiré} ;
32:     Tant que  Chemin[dernière] ≠ S_init Faire
33:         Si Chemin[dernière] ∈ Noeuds Alors
34:             k := l'indice de Chemin[dernière] dans Noeuds ;
35:             Ajouter Liste_provenance [k] à Chemin ;
36:         Si non
37:             Returner Pas de chemin ;
38:         Fin Si
39:     Fin Tant que
40: Fin Fonction
```

4.6 Les étapes de la réalisation

Au début nous avons réalisé un planificateur de chemin basé sur la PRM pour un robot ponctuel dans une scène en 2D, ensuite, nous avons développé ce planificateur pour un robot polygonal dans un environnement à 2D.

4.6.1 Robot ponctuel

Nous avons réalisé le planificateur de chemin basé sur la PRM pour cette partie suivant les étapes :

1. Le premier travail à faire consiste à programmer une fonction appelée 'CollisionCheck' qui teste la collision d'une configuration avec tous les obstacles, et pour cette étape, la fonction MATLAB *inpolygon* est utilisée.
2. Nous concevons la fonction 'GenerateSamples' qui génère n configurations libres et les sauvegarde sous forme d'un vecteur V, et ceci à l'aide de la fonction 'CollisionCheck'. Pour cette étape nous avons utilisé l'échantillonnage uniforme à l'aide de la fonction MATLAB *rand*.
3. Nous concevons la fonction 'NearestNeighbors', qui permet de trouver les adjacents d'une configuration selon k et *dist* et les sauvegarde sous forme d'un vecteur N_q. Tel que *dist* est une distance euclidienne dans cette étape.
4. Nous écrirons maintenant la fonction 'LocalePlannifier', qui réalise les connexions entre chaque configuration dans V avec ses adjacents trouvée par la fonction 'NearestNeighbors'. Nous avons utilisé pour cette partie la fonction MATLAB *polyxpoly* pour tester la collision des chemins locaux.
5. On programme maintenant la fonction 'GeneratePath', qui connecte les deux configurations q_{init} , q_{goal} à la carte, en utilisant le planificateur local précédant, puis elle cherche le plus court chemin entre ces deux en utilisant les deux méthodes : Dijkstra (Algorithme 12) et A^* (Algorithme 2), puis elle effectue le post-traitement sur le chemin obtenu en utilisant l'algorithme 5.

Après la réussite de ces étapes, nous avons programmé les deux méthodes d'échantillonnage : gaussien et par pont.

4.6.2 Robot polygonal

La PRM de ce cas est construit en utilisant la PRM précédente avec quelques changements, en considérant cette fois-ci un robot sous forme d'un polygone dans le plan (Figure 4.2), Les changements qu'il faut faire sur la procédure précédente concernent :

1. En premier lieu nous changeons la procédure du test de la collision pour une configuration donnée. Ceci revient à tester si le polygone représentant le robot n'est pas en collision avec les polygones représentant les obstacles ce qui est complètement différent des tests effectués quand le robot est un point matériel. Pour ce cas nous avons utilisé la fonction MATLAB *polybool*.
2. Le deuxième changement concerne la procédure d'échantillonnage. Le robot ayant une forme polygonale sa configuration est déterminée par les coordonnées de l'origine du repère qui lui est attaché et de son orientation (rotation) par rapport au repère de l'espace (repère fixe). Alors le test de collision a besoin d'une matrice de transformation qui permet d'exprimer la position du robot.
3. Le troisième changement se fait au niveau de planificateur local. Cette fois on utilise un planificateur qui teste la collision pour tous les points d'un chemin local, on parle alors sur des algorithmes (Algorithme 8) (Algorithme 9) (Algorithme 11).

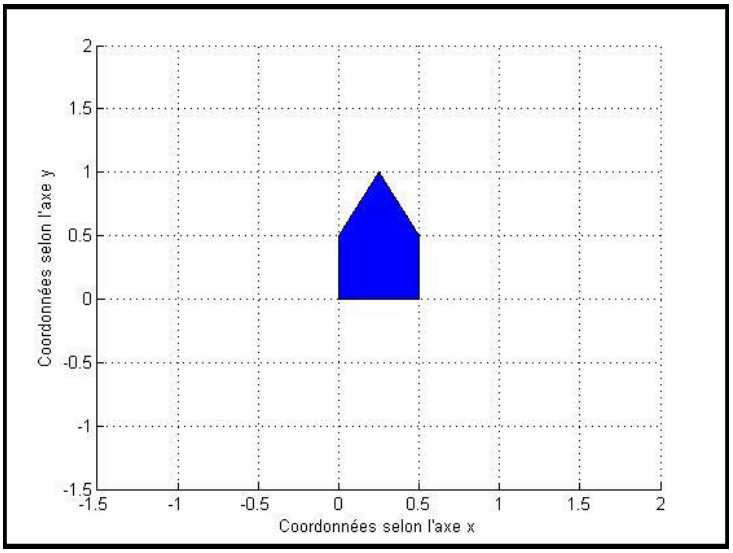

FIGURE 4.2 – Forme polygonale convexe considérée pour approximer le robot.

4.7 Simulation

Nous considérons deux scènes différentes en 2D pour tester nos algorithmes (Figure 4.3). Tel que la scène 1 est simple, dans laquelle les obstacles sont un peu éloignés les uns des autres ; et la scène 2 est difficile, telle qu'elle contient des passages étroits pour tester les algorithmes d'échantillonnage qui traitent du problème de ce genre de passages. Nous

considérons aussi que les obstacles sont des polygones non convexes, et l'environnement est statique.

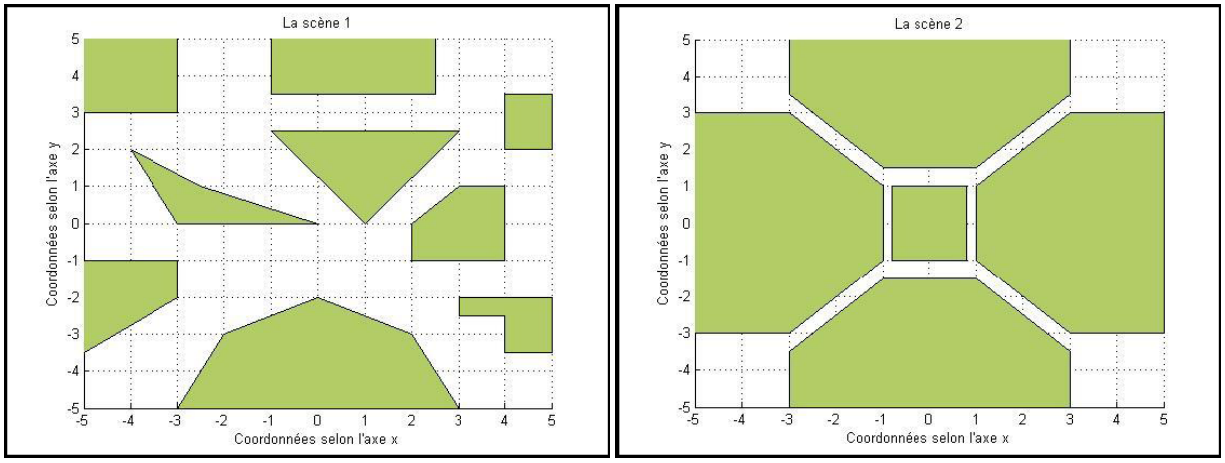

FIGURE 4.3 – Les deux scènes de simulation.

Nos simulations sont faites sous MATLAB version 2014, avec un microprocesseur Intel Core i3 2.20 Ghz. Nous avons divisé la simulation selon la forme du robot en deux parties : le robot est ponctuel et le robot est polygonal, tel que dans la première partie, nous varions les trois méthodes d'échantillonnage avec les deux scènes, et dans la deuxième partie, nous varions les deux planificateurs locaux avec la scène 1. Dans les tableaux des résultats nous allons utiliser les notations suivantes :

T_E : est le temps d'échantillonnage.

T_C : est le temps de connexion entre les échantillons.

$T_{R-D_{ij}}$,T_{R-A^*} : est le temps de recherche du plus court chemin.

NaN : pas de chemin.

4.7.1 Robot ponctuel

Pour cette partie, nous avons choisis pour les deux scènes, la distance euclidienne pour la fonction distance $\mathcal{D}$, et nous avons fixés la distance maximale pour sélectionner les adjacents d'une configuration et le nombre maximum d'adjacents à sélectionner, tel que : $k = 10$ et $dist$=2. Nous avons fait trois essais pour tous les cas dans cette partie, et chaque résultat des tableaux (Table 4.1) et (Table 4.2) est la moyenne des trois essais.

Les deux configurations initiale et finale qu'on a choisi pour cette partie sont $q_{init} = [4.5, 4.5]$ $q_{goal} = [-4.5, -4.5]$ (Figure 4.4).

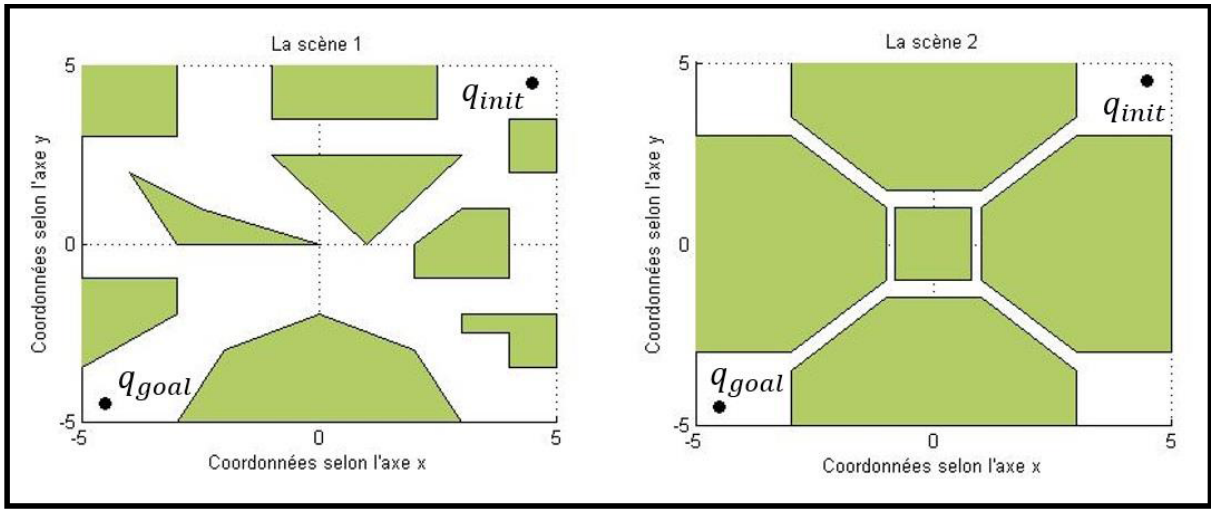

FIGURE 4.4 – La configuration initiale et la configuration finale considérées pour le robot ponctuel.

- Scène 1 :

TABLE 4.1 – Les temps d'exécutions pour le robot ponctuel dans la scène 1 (en secondes).

n	Uniforme				Gaussien				Par pont			
	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}
50	0.038	2.136	NaN	NaN	0.968	4.125	NaN	NaN	37.786	2.765	NaN	NaN
200	0.160	13.740	0.047	0.014	4.017	32.287	0.087	0.041	144.397	25.998	NaN	NaN
700	0.530	50.223	0.2	0.035	13.782	99.752	0.255	0.188	433.069	38.490	NaN	NaN

- Scène 2 :

TABLE 4.2 – Les temps d'exécutions pour le robot ponctuel dans la scène 2 (en secondes).

n	Uniforme				Gaussien				Par pont			
	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}
50	0.167	3.527	NaN	NaN	0.920	3.383	NaN	NaN	4.389	3.716	NaN	NaN
200	0.587	11.459	NaN	NaN	3.761	17.112	0.096	0.055	16.211	16.497	0.099	0.069
700	2.145	62.797	NaN	NaN	12.576	62.484	0.381	0.235	58.009	63.847	0.389	0.173

Commentaire : D'après les deux tableaux précédents on peut citer les observations suivantes :

Lorsque le nombre d'échantillons est petit ($n = 50$), le planificateur ne réussit pas à trouver le chemin avec les trois méthodes d'échantillonnage dans les deux scènes, mais avec un nombre d'échantillons suffisant ($n = 200$ et 500) le planificateur réussit avec les trois méthodes d'échantillonnage à trouver le chemin dans la scène 1. Par contre, avec la scène 2, elle ne réussit qu'avec les deux méthodes d'échantillonnage gaussien et par pont, ce qui explique que ces deux méthodes échantillonnent dans les passages étroits, et c'est donc là l'avantage de ces deux méthodes.

De la comparaison entre les temps d'échantillonnage, on remarque que la méthode d'échantillonnage uniforme est la plus rapide parmi les trois méthodes avec les deux scènes, car cette méthode génère à chaque fois une seule configuration et elle l'ajoute à la carte si elle n'est pas en collision. Par contre, les deux autres méthodes testent la collision pour deux configurations à chaque itération, ce qui prend un peu de temps, et encore ces deux méthodes rejettent les deux configurations si elles ne vérifient pas les conditions, ce qui perd aussi plus de temps.

Dans tous les cas où le planificateur réussit à trouver le chemin, on voit que l'algorithme A^* est plus rapide que l'algorithme de Dijkstra, car l'algorithme de Dijkstra calcule les plus courts chemins entre q_{init} et toutes les autres configurations, alors il doit visiter tous les sommets du graphe G. Par contre, l'algorithme A^* cherche le plus court chemin seulement entre q_{init} et q_{goal}, et il utilise l'heuristique pour guider sa recherche, alors, il ne doit pas visiter tous les sommets du graphe G, et c'est donc là l'avantage de cet algorithme.

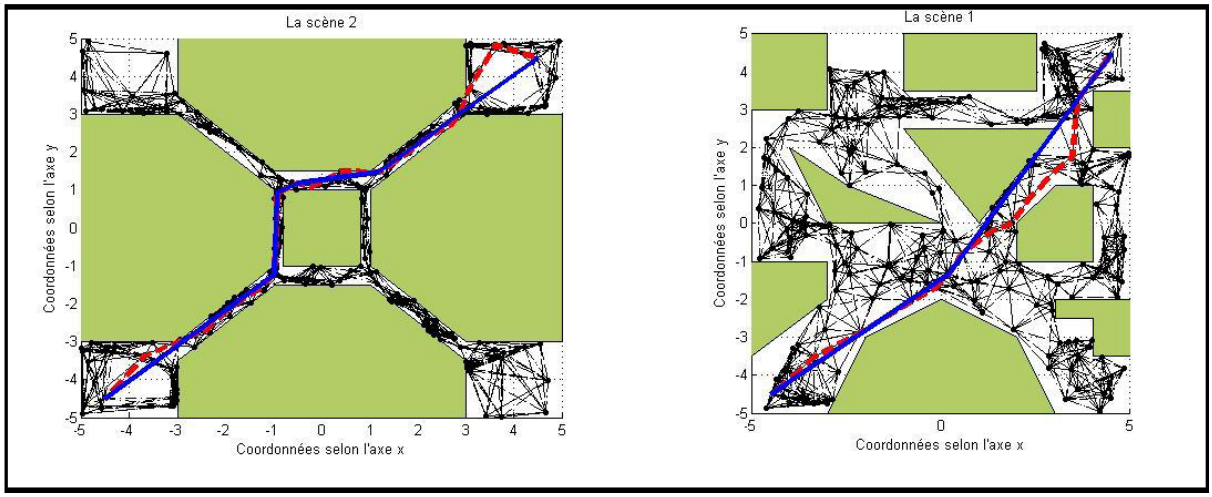

FIGURE 4.5 – Le plus court chemin dans les deux scènes. La ligne rouge discontinue est le plus court chemin dans la carte, et la ligne bleu continue est le chemin après le post-traitement. Ces deux cartes construits en utilisant l'échantillonnage uniforme pour la scène 1, et en utilisant l'échantillonnage gaussien pour la scène 2.

4.7.2 Robot polygonal

Pour cette partie, nous avons choisis la distance de Manhattan pour la fonction distance $\mathcal{D}$, et nous avons fixés la distance maximale pour sélectionner les adjacents d'une configuration et le nombre maximum d'adjacents à sélectionner, tel que :$n = 10$ et $dist = 3$. Les résultats des tableaux (Table 4.3) et (Table 4.4) sont obtenus par un seul essai.

Les deux configurations initiale et finale qu'on a choisies pour cette partie sont $q_{init} = [4, 4, \frac{\pi}{2}]$ $q_{goal} = [-3.5, -3.5, \frac{\pi}{2}]$ (Figure 4.6).

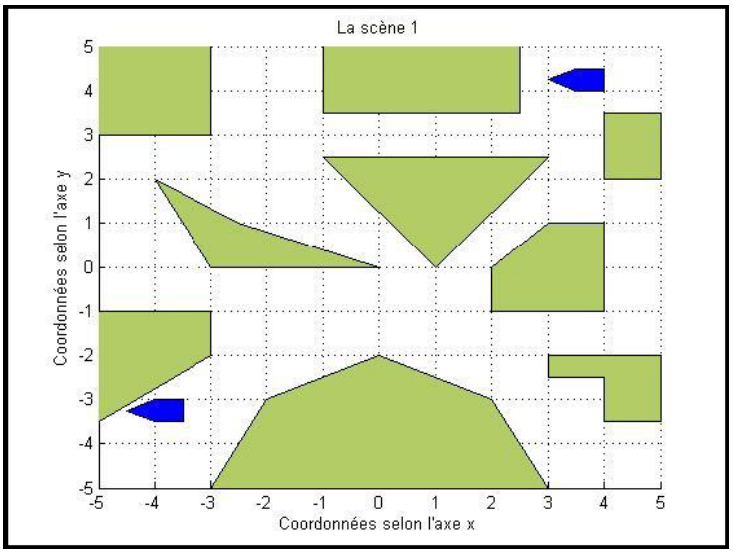

FIGURE 4.6 – La configuration initiale et la configuration finale considérées pour le robot polygonal.

- Planificateur de ligne droite :

TABLE 4.3 – Les temps d'exécutions pour le robot polygonal avec le planificateur de ligne droite (en secondes).

n	Uniforme				Gaussien				Par pont			
	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}
60	3.022	140.394	NaN	NaN	5.390	39.911	NaN	NaN	37.861	30.661	NaN	NaN
300	13.064	806.111	NaN	NaN	32.703	513.342	0.061	0.019	117.982	496.801	0.120	0.048
800	18.669	1727.461	0.283	0.065	83.912	1485.991	0.2452	0.043	656.116	2647.113	0.454	0.174

- Planificateur de rotation-à-s :

TABLE 4.4 – Les temps d'exécutions pour le robot polygonal avec le planificateur Rotation-à-s (en secondes).

n	Uniforme				Gaussien				Par pont			
	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}	T_E	T_C	$T_{R-D_{ij}}$	T_{R-A^*}
60	2.544	115.762	NaN	NaN	6.095	40.076	NaN	NaN	20.847	34.992	NaN	NaN
300	6.585	644.127	0.066	0.035	32.072	563.490	0.048	0.025	133.419	455.612	0.042	0.029
800	33.183	1760.246	0.237	0.045	87.046	1582.48	0.247	0.019	318.046	1276.249	0.171	0.035

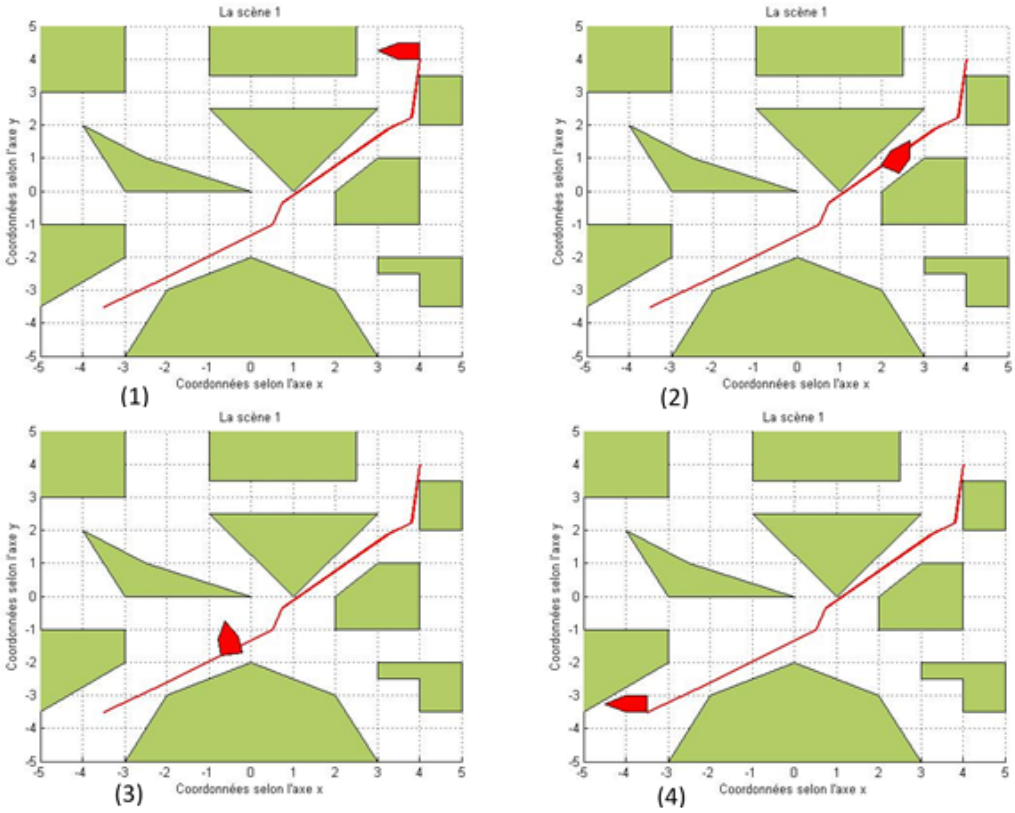

FIGURE 4.7 – Les étapes d'exécution du chemin pour le robot polygonal en utilisant le planificateur local de ligne droite.

Commentaire : D'après les deux tableaux précédents on peut citer les observations suivantes :

Avec les trois méthodes d'échantillonnage et pour les deux planificateurs locaux, le planificateur ne réussit pas à trouver le chemin lorsque le nombre d'échantillons est petit ($n = 60$), mais le planificateur réussit pour un nombre suffisant ($n = 300, 800$) avec les deux planificateurs locaux.

L'échantillonnage uniforme reste toujours le plus rapide parmi les autres.

Les performances des deux planificateurs locaux sont presque similaires pour les trois méthodes d'échantillonnage, avec un petit avantage du planificateur de rotation-à-s pour l'échantillonnage uniforme et par pont. Cette difficulté de comparaison entre les performances des deux planificateurs locaux est causée par la convergence des résultats obtenus par un seul essai. Ainsi, pour une bonne comparaison il faut beaucoup d'essais, ce qui requière un long temps et un matériels puissant.

4.8 Conclusion

Nous avons présenté dans ce chapitre les étapes de notre réalisation d'un planificateur de chemin basé sur une PRM pour un robot modélisé par un polygone dans un environnement en 2D, et nous avons présenté nos modifications sur quelques algorithmes, ensuite, nous avons fait quelques comparaisons entre les performances des méthodes présentés dans le chapitre 2 et 3 que nous avons implémenté.

Conclusion générale

Nous avons présenté dans ce mémoire le principe de fonctionnement des planificateurs de mouvement basé sur une PRM, et quelques méthodes proposées dans les littérateurs concernant l'échantillonnage et la planification local des chemins, et nous avons également présenté des outils importants pour construire une PRM, tel que le test de collision et la recherche du plus court chemin, où nous avons présenté une méthode très efficace pour chercher le plus court chemin dans un graphe, dans laquelle elle utilise l'heuristique pour guider sa recherche. Ensuite, nous avons abordé les étapes de notre réalisation d'un planificateur de chemin basé sur la carte de route probabiliste pour un robot modélisé par un polygone dans un environnement en 2D.

Notre planificateur ne traite que les cas avec des scènes en 2D, mais il reste un travail important et une base pour réaliser un autre planificateur avec une scène en 3D. La plupart du temps passé dans la construction d'une PRM est avec la procédure de test de collision. Le principe commun de cette procédure est de tester la collision d'une configuration donnée avec tous les obstacles existant dans la scène, alors le temps exigé pour le test de collision est proportionnel avec le nombre des obstacles. Une amélioration consiste à réduire le nombre de tests en utilisant l'heuristique pour localiser le robot par exemple.

Les extensions futures de notre travail consistent à développer ce planificateur pour un robot modélisé par un polyèdre dans un environnement en 3D, ensuite de développer ce planificateur pour des robots manipulateurs en traitant d'abord le cas des manipulateurs planaires. Une autre extension consiste à prendre en considération des obstacles dynamiques à éviter.

Bibliographie

[1] Lozano-Perez, Tomas,*Spatial planning : A configuration space approach*,Computers, IEEE Transactions on 100.2 (1983) : 108-120.

[2] Kavraki, Lydia E., et al, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*,Robotics and Automation, IEEE Transactions on 12.4 (1996) : 566-580.

[3] Svestka, Petr, *On probabilistic completeness and expected complexity of probabilistic path planning*, Universiteit Utrecht, Faculty of Mathematics and Computer Science, 1996.

[4] Latombe, Jean-Claude, *Motion planning : A journey of robots, molecules, digital actors, and other artifacts*,The International Journal of Robotics Research 18.11 (1999) : 1119-1128.

[5] Craig, John J, *Introduction to Robotics Mechanics and control*, Vol. 3. Upper Saddle River : Pearson Prentice Hall, 2005.

[6] Spong, M.W., Hutchinson, S., and Vidyasagar, M., *Robot Modeling and Control*, John Wiley and Sons, 2006.

[7] Kurfess, Thomas R., ed, *Robotics and Automation Handbook*,CRC press, 2004.

[8] W. Khalil, E. Dombre, *Modélisation, identification et commande des robots*, Hermès, 1999.

[9] Choset, Howie M, *Principles of Robot Motion Theory, Algorithms, and Implementations*, MIT press, 2005.

[10] S. M. R. Rahman, *Performance of local planners with respect to sampling strategies in sampling-motion planner*, McGill University, master thesis, June 21, 2010.

[11] Acar, Ercan U., et al, *Morse decompositions for coverage tasks*, The International Journal of Robotics Research 21.4 (2002) : 331-344.

[12] Guibas, Leonidas J., et al, *Visibility-based pursuit-evasion in a polygonal environment*, Algorithms and Data Structures. Springer Berlin Heidelberg, 1997. 17-30.

[13] Khatib, Oussama, *Real-time obstacle avoidance for manipulators and mobile robots*, The international journal of robotics research 5.1 (1986) : 90-98.

[14] Gondran .M, Minoux Michel, *Graphes et algorithmes*, 4 édition, Lavoisier, 2009.

[15] Dijkstra, Edsger W, *A note on two problems in connexion with graphs*, Numerische mathematik 1.1 (1959) : 269-271.

[16] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael, *A formal basis for the heuristic determination of minimum cost paths*, Systems Science and Cybernetics, IEEE Transactions on 4.2 (1968) : 100-107.

[17] LaValle, Steven M, *Planning Algorithms*, Cambridge University Press, 2006.

[18] Christer E, *Real-Time Collision Detection*, CRC Press, 2004.

[19] Hormann, Kai, and Alexander Agathos, *The point in polygon problem for arbitrary polygons*, Computational Geometry 20.3 (2001) : 131-144.

[20] Jiménez, Juan José, Rafael J. Segura, and Francisco R. Feito, *Efficient collision detection between 2D polygons*, (2004).

[21] Badawy, Wael M., and Walid G. Aref. , *On Local Heuristics to Speed Up Polygon-Polygon Intersection Tests*, Proceedings of the 7th ACM international symposium on Advances in geographic information systems. ACM, 1999.

[22] Kavraki, Lydia E, *Random networks in configuration space for fast path planning*, Diss. stanford university, 1994.

[23] Sheldon M. Ross, *Introduction to Probability Models*, Elsevier, 2014.

[24] Kuffner, James J, *Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning,*Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on. Vol. 4. IEEE, 2004.

[25] Boor, Valdrie, et al, *The Gaussian sampling strategy for probabilistic roadmap planners*, Robotics and automation, 1999. proceedings. 1999 ieee international conference on. Vol. 2. IEEE, 1999.

[26] Hsu, David, et al, *The bridge test for sampling narrow passages with probabilistic roadmap planners*, Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on. Vol. 3. IEEE, 2003.

[27] Amato, Nancy M., et al, *Choosing good distance metrics and local planners for probabilistic roadmap methods*, Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on. Vol. 1. IEEE, 1998.

Résumé

L'objectif de ce mémoire est de réaliser sous MATLAB, un planificateur de chemin basé sur la carte de route probabiliste pour un robot modélisé par un polygone dans un environnement statique en 2D encombré d'obstacles sous forme des polygones non convexes, et ceci en implémentant trois méthodes d'échantillonnage : l'échantillonnage uniforme, l'échantillonnage gaussien et l'échantillonnage par pont, tel que les deux dernières traitent le problème de passage étroit. Nous avons aussi implémenté deux planificateurs locaux : le planificateur de ligne droite et le planificateur de rotation-à-s. Et pour trouver le plus court chemin entre la configuration initiale et la configuration finale, nous avons utilisé deux algorithmes de recherche le plus court chemin dans un graphe : Dijstra et A^*. Nous avons simulé notre planificateur avec deux scènes différentes.

Mots-clés : planificateur de chemin, carte de route probabiliste, méthode d'échantillonnage, planificateur locale, configuration.

Abstract

The objective of this thesis is to realize in MATLAB a path-planner based on the probabilistic roadmap for a robot modeled by polygon in a static environment which is cluttered with obstacles in the form of non-convex polygons, where we have used three sampling methods : uniform sampling, Gaussian sampling and bridge sampling, where the two last methods address the narrow passage problem. We have also implemented two locales planners : straight-line planner, rotate-at-s planner. And to find the shortest path between the initial configuration and the goal configuration, we used two shortest path search algorithms in a graph : Dijksta and A^*. We simulated our path-planner with two different scenes.

Keywords : path-planner, probabilistic roadmap, sampling method, locale planner, configuration.