THESE

présentée par

Juan Manuel AHUACTZIN LARIOS

pour obtenir le titre de DOCTEUR

de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Spécialité : INFORMATIQUE

Thèse preparée au sein du
Laboratoire d'**I**nformatiqe **F**ondamentale et d'**I**ntelligence **A**rtificielle

LE FIL D'ARIANE : UNE METHODE DE PLANIFICATION GENERALE. APPLICATION A LA PLANIFICATION AUTOMATIQUE DE TRAJECTOIRES.

Date de la soutenance : 29 septembre 1994

Composition du jury :

Président :	Philippe CINQUIN
Rapporteurs :	Etienne DOMBRE
	Jean-Paul LAUMOND
Examinateurs :	Ofelia CERVANTES
	Jacques POT
Directeur de thèse :	Emmanuel MAZER

Remerciements

*“ Laissez-moi commencer par un mot que tous les hommes, depuis que l'homme existe, ont prononcé : **merci**”*[1]

Ainsi, avec cette merveilleuse parole, je remercie les membres du jury pour avoir accepté de juger ce travail et pour le temps qu'ils ont consacré à la révision de celui-ci.

Je voudrais adresser mes remerciements à Monsieur *Philippe CINQUIN,* Professeur à l'Université Joseph Fourier, qui nous a fait l'honneur de présider le jury de cette thèse.

Je remercie également Monsieur *Etienne DOMBRE*, Directeur de Recherche CNRS au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, ainsi que Monsieur *Jean-Paul LAUMOND*, Directeur de Recherche CNRS au Laboratoire d'Automatique et d'Analyse des Systèmes du CNRS à Toulouse, pour avoir accepté de juger ce travail, et pour le temps qu'ils ont consacré à l'élaboration des rapports de celui-ci.

Je tiens à remercier Madame *Ofelia CERVANTES*, Doyenne de l'Ecole d'Ingénieurs de l'Université *“Universidad de las Américas, Puebla”* au Mexique, pour le soutien et l'intérêt qu'elle a toujours porté pour ce travail, ainsi que pour avoir accepté de faire partie du jury de ma thèse.

J'adresse également mes remerciements à Monsieur *Jacques POT,* Responsable du département de robotique à EDF Paris, pour l'intérêt qu'il a montré pour ce travail et pour l'honneur qu'il m'a fait de faire parti du jury de ma thèse.

Finalement, dans le jury de cette thèse, je tiens à remercier profondément mon directeur de thèse, Monsieur *Emmanuel MAZER*, Chargé de Recherche CNRS, pour toutes les connaissances scientifiques qu'il a partagées avec moi, pour sa confiance, ses conseils, et pour l'enthousiasme qu'il a toujours porté pour ce travail. Merci “manu” et que cette thèse soit le début d'autres travaux que l'on réalisera ensemble dans l'avenir.

Je remercie particulièrement Monsieur *Pierre BESSIERE* pour tous ses conseils et pour l'enthousiasme qu'il a toujours porté à ce travail. J'ai vraiment apprécié sa disponibilité et sa manière de travailler. Sans ses inestimables apports, ce travail ne serait pas le même.

Je voudrais aussi remercier Monsieur *Philippe JORRAND,* Directeur du LIFIA, pour m'avoir accepté dans son laboratoire.

Un grand merci aussi à tous le membres de l'équipe SHARP avec qui j'ai partagé

[1] *Octavio PAZ (écrivant mexicain), discours de Stockholm, “La quête du présent”*

de très agréables moments de travail et de détente, très particulièrement à *Christian BARD* (pour le support technique) , *Moëz CHERIF, Fernando DE LA ROSA, Philippe GARNIER, Mouna HASSOUN* et au responsable de l'équipe *Christian LAUGIER* Directeur de Recherche INRIA.

Un autre grand merci à mes collègues étudiants sous la direction de Emmanuel MAZER: *Remis BALANIUK, Eric DEDIEU et Olivier LEBELTEL.* A eux tous je voudrais exprimer mes remerciements pour les idées et suggestions apportées à cette thèse mais surtout, pour leur amitié.

Ce travail n'aurait pas été possible sans la collaboration du LGI que je remercie profondément pour sa contribution. En particulier, je voudrais adresser mes remerciements à *Thierry CHATROUX, El-Gazali TALBI et Traian MUNTEAN* avec qui j'ai eu le plaisir de travailler pendant le développement d'une grande partie pratique de cette thèse.

Mes chaleureux remerciements à *Mai PHAM,* qui à gentiment lu ma thèse en apportant à chaque fois des suggestions très avisées.

Il me faudrait une liste énorme pour remercier tous mes amis mexicains et français qui mont encouragé tout au long de mes études. Ami, trouve dans cette ligne tous mes remerciements.

Finalement, je remercie le Conseil National de Science et Technologie du Mexique (CONACYT) et l'Université *"Universidad de las América, Puebla"* pour son soutien sans lequel ce travail n'aurait pu être accompli.

Juan Manuel AHUACTZIN-LARIOS
Septembre 1994

Sommaire

1 Introduction **15**

1.1 Présentation du problème 15

1.2 Principe de la méthode proposée 16

1.3 Résultats obtenus 20

1.4 Introduction à l'algorithme Fil d'Ariane 21

1.4.1 Présentation de l'algorithme 21

1.4.2 L'algorithme SEARCH 25

1.4.3 L'algorithme EXPLORE 26

1.5 Organisation du document 28

1.5.1 Partie I (Le Fil d'Ariane: un algorithme général de planification) 29

1.5.2 Partie II (Application au problème de planification en robotique) 29

1.6 Cadre de l'étude 31

1.7 Travaux proches de notre méthode 31

1.7.1 La méthode stochastique 31

1.7.2 La construction incrémentale 32

I Le Fil d'Ariane : Un algorithme de planification général 35

2 Définitions 39

2.1 L'espace des configurations 39

2.2 Chemins et ensembles chemin-connectés 42

2.3 La distance de Hausdorff 43

2.4 Billage et pavage d'un espace 44

3 L'algorithme $EXPLORE$ 51

3.1 L'algorithme $EXPLORE_{\infty}$ 51

3.2 Espace libre à ε-près et chemins Manhattan 55

3.2.1 L'espace libre à $\varepsilon-$près 55

3.2.2 Les Chemins Manhattan 55

3.3 L'algorithme $EXPLORE$ 60

3.3.1 Description de l'algorithme 60

4 Améliorations de l'algorithme $EXPLORE$ 67

4.1 L'algorithme $SEARCH$ 67

4.1.1 Définition 67

4.1.2 La "pré-image Manhattan" d'ordre 1 68

4.2 Génération efficace de chemins Manhattan en C_{libre} 70

4.2.1 L'ensemble propre d'une trajectoire Manhattan 70

4.2.2 Le "rebondissement" des trajectoires 71

4.3 L'algorithme Fil d'Ariane 73

II Application au problème de planification de trajectoires de robots 75

5 Les Algorithmes Génétiques **77**

5.1 Rappel des problèmes d'optimisation à traiter 78

5.2 Introduction aux algorithmes génétiques 78

5.2.1 Exemple : Le problème de la fonction cinématique inverse . . 80

5.3 La parallélisation des algorithmes génétiques 84

5.3.1 Modèle parallèle standard 84

5.3.2 Modèle à décomposition 85

5.3.3 Modèle massivement parallèle (PGA) 86

5.3.4 Evaluation des performances de l'algorithme 89

5.4 Utilisation de l'algorithme PGA dans l'algorithme Fil d'Ariane . . . 91

5.5 Pourquoi avoir choisi les algorithmes génétiques ? 91

6 Planification de trajectoires pour un robot mobile holonome **93**

6.1 Description de l'implantation et résultats expérimentaux 94

6.1.1 Implantation de l'algorithme $SEARCH$ 94

6.1.2 Implantation de l'algorithme $EXPLORE$ 96

6.1.3 La combinaison des deux fonctions 97

6.2 Discussion sur le comportement de l'algorithme $EXPLORE$ 100

6.2.1 La décroissance de la fonction $EXPLORE(t)$ 100

6.2.2 Informations obtenues par EXPLORE(t) 103

6.2.3 Où ont été placées les balises ? 104

7 Planification de trajectoires pour un robot à six degrés de liberté **105**

7.1 Description générale du système 106

7.1.1 Utilisation du système 108

7.2 Le système de modélisation 108

7.2.1 Le premier niveau de modélisation et le système ACT 109

7.2.2 Création du modèle optimisé (deuxième niveau) 110

7.2.3 Le modèle sphérique (troisième niveau) 114

7.3 Le calcul de débattements 114

7.3.1 Calcul de débattements entre deux parallélépipèdes 115

7.3.2 Mise à jour de la position du robot et des obstacles 118

7.3.3 Transformations homogènes des obstacles dans les repères des axes de rotation 123

7.4 Le codage et décodage des trajectoires 123

7.4.1 Le codage des domaines de recherche de $SEARCH$ et $EXPLORE$ 123

7.4.2 Le décodage d'une trajectoire 124

7.4.3 Reconstruction de la trajectoire finale 126

7.5 Discussion 132

7.5.1 La difficulté de trouver une solution 132

7.5.2 Les calculs requis pour une trajectoire Manhattan et les calculs filtrés 134

7.5.3 Le filtrage des calculs 134

8 Implantation parallèle de l'algorithme Fil d'Ariane 137

8.1 Les Niveaux de parallélisation 138

8.1.1 Le premier niveau de parallélisation 138

8.1.2 Le deuxième niveau de parallélisation 140

8.1.3 Troisième niveau de parallélisation 141

8.2 L'implantation physique 144

8.2.1 Organisation physique du premier et du deuxième niveau de parallélisation 146

8.2.2 Organisation physique du troisième niveau de parallélisation 146

9 Perspectives 151

9.1 Discussion et direction des recherches 151

9.1.1 L'algorithme $EXPLORE$ et les diagrammes de Voronoi . . 151

9.1.2 Utilisation de l'information donnée par $EXPLORE(t)$ . . . 154
9.1.3 Etude sur le rebondissement 154
9.1.4 Etude sur l'ordre des trajectoires 155
9.1.5 Changement de technique d'optimisation 156
9.1.6 Utilisation d'un modèle hiérarchique 157
9.1.7 Utilisation d'autres techniques de génération de trajectoires 157
9.2 Conclusions 157
9.2.1 Sur le plan théorique 157
9.2.2 Sur le plan expérimental 158
9.2.3 Discussion 159

A Synthèse des travaux existant en planification de trajectoires 163
A.0.4 Classification des problèmes par type d'environnement . . . 163
A.0.5 Classification des problèmes par type de robot 165
A.1 Classification des approches de la planification 166
A.1.1 Classification par les résultats 166
A.1.2 Classification par type d'approche 166
A.1.3 Complexité du problème 172

B Calculs Géométriques 175
B.1 Entités Géométriques 176
B.2 Calculs de débattement 177

Liste des Figures

1.1 Suite des déplacements effectués à partir du décodage de $\hat{x}$. 17

1.2 Séquence de solutions trouvées par un algorithme d'optimisation. . 19

1.3 Un problème avec un minimum local. 19

1.4 Construction de l'arbre des balises. 20

1.5 L'espace de travail d'un robot planaire à deux degrés de liberté. . . 23

1.6 La "pré-image" de $\hat{q}_\bullet$ produite par le planificateur L. 24

1.7 La "post-image" de EL produite par le planificateur L. 25

2.1 L'espace de travail d'un robot planaire à deux degrés de liberté. . . 41

2.2 L'espace des configurations et ses composantes. 41

2.3 Convexité d'un espace. 45

2.4 Deux billages de densité différente en $\mathbb{R}^2$. 46

2.5 Un simplex régulier en $\mathbb{E}^2$ et un autre en $\mathbb{E}^3$. 47

2.6 Billage d'un espace X et le cube $C_{s(X)}$. 48

3.1 L'espace C^{ε}_{libre}. 56

3.2 Une fonction $\hat{\varphi} \in \wp(2)$ et une autre $\hat{\gamma} \in M(2)$ 58
3.3 Image de $\hat{\varphi}$ et $\hat{\gamma}$. 59
3.4 Les fonctions γ_1 et γ_2. 59
3.5 Un chemin Manhattan dans l'espace des configurations. 61
3.6 Pavage de C_{libre}. 64

4.1 La pré-image de deux points en C_{libre}. 69
4.2 Une trajectoire où $\hat{\gamma}(1) \in P_M(\hat{q}_\bullet)$. 70
4.3 Le concept de rebondissement. 71
4.4 Le segment A_m calculé à partir d'une configuration $\hat{q}_{m-1}$. 72

5.1 Etapes d'un algorithme génétique. 79
5.2 Mutation d'un individu. 80
5.3 Combinaison des individus. 80
5.4 Une configuration du bras manipulateur et son codage. 81
5.5 Graphique de la fonction d'inversion de coordonnées. 82
5.6 Distribution de la population initiale. 83
5.7 Convergence de la population. 84
5.8 Modèle parallèle standard. 85
5.9 Modèle parallèle à décomposition. 86
5.10 Un tore de 16 processeurs. 88
5.11 Accélération de l'algorithme génétique parallèle. 89
5.12 Qualité de la solution en fonction de la taille de la population. . . . 90

6.1 Codage des trajectoires pour le robot holonome et le rebondissement. 94
6.2 La pré-image des mouvements "directs" au but et la pré-image P_L. 95
6.3 Deux chemins trouvés par $SEARCH$. 96
6.4 Planification de trajectoires pour le robot holonome dans un environnement dynamique. 97

6.5 La post-image produite par une balise. 98
6.6 L'évolution des balises dans l'espace libre. 98
6.7 Le "fil d'Ariane" des balises pour le robot mobile holonome. 99
6.8 Deux trajectoires planifiées par l'algorithme Fil d'Ariane. 100
6.9 L'espace à Explorer. 101
6.10 Graphiques de $EXPLORE(t)$. 102
6.11 L'environnement exploré et le graphique $EXPLORE(t)$ obtenu. . . 103
6.12 L'écart de L_t. 104

7.1 Architecture du système. 107
7.2 Scène réelle du site expérimental du LIFIA. 109
7.3 Premier niveau de modélisation. 110
7.4 Deuxième niveau de modélisation. 111
7.5 La chaîne articulaire du robot et les matrices de transformation. . . 112
7.6 Décomposition d'un parallélépipède. 113
7.7 Equivalence des *valeurs* de $\psi^{-1}(x,y)$ et $\psi(y,x)$. 115
7.8 Contact type A. 116
7.9 Contact type B. 117
7.10 Contact type C. 118
7.11 Etapes du calcul de la matrice M. 121
7.12 Mise à jour de la matrice M. 122
7.13 Processus de décodage d'une trajectoire. 126
7.14 L'arbre de balises. 127
7.15 Balises L_1 et L_2. 128
7.16 Balises L_3 et L_4. 129
7.17 Balises L_5 et $\hat{q}_\bullet$. 130
7.18 Simplification. 132

7.19 Un exemple de planification fictif. 133
7.20 Nombre de calculs élémentaires requis pour une trajectoire Manhattan d'ordre 1. 135

8.1 Niveaux de parallélisation de l'Algorithme Fil d'Ariane. 145
8.2 Configuration du réseau de Transputers. 147
8.3 Décomposition d'un parallélépipède. 148
8.4 Répartition des calculs de type A sur les processeurs de la "ferme". 148
8.5 Répartition des calculs de type B sur les processeurs de la "ferme". 149
8.6 Répartition des calculs de type C sur les processeurs de la "ferme" . 149

9.1 Un ensemble de balises et les polygones engendrés. 152
9.2 Espace dicrétisé et les chaînes de Markov. 155
9.3 Chaîne de Markov. 156

A.1 Graphe de visibilité. 169
A.2 Graphe de Voronoi. 169
A.3 Décomposition exacte. 170
A.4 Décomposition approximative. 170
A.5 Exemple de l'approche potentielle. 172

Chapitre 1

Introduction

1.1 Présentation du problème

Dans cette thèse nous abordons le problème de la *planification automatique de trajectoires en robotique*. Pour résoudre ce problème il faut trouver une suite de mouvements valides qui permettent à un robot de passer d'un état initial à un état final désiré. D'une manière générale, un mouvement valide est un déplacement qui ne produit pas de collisions et qui respecte les contraintes cinématiques du robot.

Si ce problème semble relativement facile à résoudre pour un opérateur, la détermination automatique d'une telle trajectoire est en fait un problème très complexe.

Depuis une vingtaine d'années de nombreuses techniques de planification automatique de trajectoires ont été proposées; néanmoins aucune ne s'est véritablement imposée comme une méthode générale pouvant résoudre de façon satisfaisante le problème, notamment au niveau industriel. En première approximation, on peut distinguer deux types de méthodes : celles qui cherchent à construire une représentation de l'espace libre (méthodes globales) et celles qui se basent sur des informations locales pour construire incrémentalement une trajectoire (méthodes locales).

L'avantage de la première approche est la complétude : une solution sera trouvée s'il en existe une. L'inconvénient est le temps de calcul qui peut croître de façon exponentielle avec le nombre de degrés de liberté; ce qui limite en pratique leurs utilisations. Inversement, les méthodes dites "locales" permettent de résoudre dans un temps raisonnable des problèmes plus réalistes au détriment de la complétude.

Une autre caractéristique de la planification de trajectoires est la difficulté de distinguer, d'une manière automatique, les problèmes faciles des problèmes difficiles. La difficulté du problème n'est pas alors considérée pour chercher une solution et donc, le temps de calcul est souvent indépendant de celle-ci.

Dans ce cadre, nous nous intéressons à définir un planificateur qui garantisse la complétude à une résolution donnée et qui adapte naturellement son temps d'exécution à la difficulté des problèmes posés.

1.2 Principe de la méthode proposée

Comme nous l'avons signalé précédemment, la plupart des planificateurs globaux procèdent par construction et exploration de l'espace des configurations $\mathcal{C}$. Dans $\mathcal{C}$, l'état du robot, ou configuration, est entièrement déterminé par un point, et les positions physiquement impossibles à atteindre sont représentées par des sous-espaces interdits de cet espace appelés $\mathcal{C}$-*obstacles*. Le problème de la planification revient alors à trouver un chemin d'une configuration initiale $\hat{q}_\circ$ à une configuration finale $\hat{q}_\bullet$ qui ne passe par aucun des $\mathcal{C}$-*obstacles*, c'est-à-dire un chemin dans l'espace $\mathcal{C}_{libre}$ défini comme le complémentaire des $\mathcal{C}$-*obstacles* dans $\mathcal{C}$.

Contrairement à la plupart de ces planificateurs, le principe de notre méthode n'est pas basé sur l'espace des configurations mais sur **l'espace des commandes**. On définit un plan comme un ensemble de commandes pour les actionneurs.

Par exemple, la trajectoire d'un robot mobile holonome peut être le résultat d'un nombre fini de commandes de déplacement et de rotation, c'est-à-dire qu'elle peut être décrite par un plan donné sous la forme $(\theta_1, l_1, \theta_2, l_2, \dots \theta_\ell, l_{ell})$ où cette séquence est interprétée comme : tourner de θ_1 puis avancer de l_1, tourner de θ_2, etc...

La figure 1.1 montre les configurations intermédiaires et la configuration terminale $\hat{q}_\ell$ obtenues pour un robot holonome après le décodage du plan :

$$\hat{x} = (\theta_1, l_1, \theta_2, l_2, \theta_3, l_3)$$

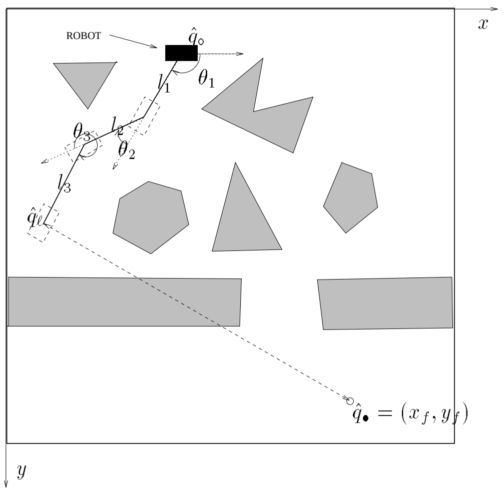


FIG. 1.1 - Suite des déplacements effectués à partir du décodage de $\hat{x}$.

Un autre cas envisageable est celui d'une voiture : les commandes réalisables peuvent, par exemple, contrôler deux actionneurs : l'accélérateur et l'angle du volant. De cette façon, une trajectoire pour le véhicule pourrait être décrite par une suite

$$(a_1, \theta_1, a_2, \theta_2, \ldots, a_{ell}, \theta_{ell})$$

où a_i nous indique l'accélération ou décélération et θ_i la variation de l'angle du volant. Les valeurs a_i et θ_i pourraient, par exemple, être appliquées pendant un temps constant Δt.

Ainsi, si un robot dispose de n commandes différentes pour se déplacer, nous pouvons obtenir, à partir d'un vecteur $\hat{x} = (x_1, \ldots, x_k) \in I\!\!R^k$ où $k \leq n$, une suite d'actions du robot et par conséquent une trajectoire.

Une fois établies les actions produites pour chacun des $x_i \in \hat{x}$, on peut, à partir d'un vecteur $\hat{x} \in I\!\!R^k$ et de la configuration initiale du robot $\hat{q}_\circ$, obtenir une configuration terminale. Cette configuration terminale est le résultat de l'application des commandes codées par $\hat{x}$. De même si l'on exécute les commandes codées par ℓ vecteurs de $I\!\!R^k$, c'est-à-dire par un vecteur $\hat{x} = (\hat{x}_1, \hat{x}_2, .., \hat{x}_\ell) \in I\!\!R^{k*\ell}$, on obtient une configuration terminale et $(\ell - 1)$ configurations intermédiaires. Ces configurations intermédiaires sont les configurations résultant des commandes codées par chacun des vecteurs $\hat{x}_i$ pour $i \in [1, 2, \ldots, \ell - 1]$. Etant donnée la configuration initiale $\hat{q}_\circ$ on obtient la configuration $\hat{q}_1$ lorsqu'on applique les commandes codées par $\hat{x}_1$, puis on obtient $\hat{q}_2$ à partir de $\hat{x}_2$, etc. La configuration terminale est alors $\hat{q}_\ell$.

Notre méthode se base sur l'optimisation d'une fonctionnelle sur cet espace de plans. Etant donnée une configuration finale $\hat{q}_\bullet$, nous définissons la fonction F : $I\!R^{n*\ell} \rightarrow I\!R^+$ comme la distance entre la configuration terminale $\hat{q}_\ell$ produite par $\hat{x}$ et la configuration finale désirée $\hat{q}_\bullet$.

$$F(\hat{x}) = ||\hat{q}_\ell - \hat{q}_\bullet||$$

S'il existe un plan $\hat{x}^\star$ qui nous amène à $\hat{q}_\bullet$ alors $F(\hat{x}^\star) = 0$. F **étant positive, chercher un plan revient donc à minimiser** F **sur** $I\!R^{k*\ell}$**.**

Par exemple, dans le cas du robot mobile, si nous ne considérons pas l'orientation finale du robot, la fonction F est la distance Euclidienne entre la position (x, y) de $\hat{q}_\ell$ et la position du but (x_f, y_f) (voir figure 1.1).

Bien entendu, il existe des vecteurs dans $I\!R^{k*\ell}$ qui codent des plans invalides comme ceux dont la trajectoire résultante passe par les obstacles. Dans ce cas on definit que la configuration terminale de $\hat{x}$ est en fait la dernière configuration intermédiaire avant que la collision ne se produise. On peut alors réécrire F de la manière suivante :

$$F(\hat{x}) = ||\hat{q}_i - \hat{q}_\bullet||$$

où $\hat{q}_i$ est la dernière configuration avant qu'une collision ne se produise.

Lorsqu'une méthode d'optimisation est utilisée pour minimiser la fonction F dans l'espace $I\!R^{n*\ell}$, on obtient une suite de solutions $(\hat{y}^1, \hat{y}^2, \ldots \hat{y}^m)(\hat{y}^i \in I\!R^{k*\ell})$ qui s'améliorent avec le temps, autrement dit qui approchent de plus en plus le robot de la configuration finale $(F(\hat{y}^1) \geq F(\hat{y}^2) \geq \ldots F(\hat{y}^m))$. Dans la figure 1.2 on montre une suite de solutions obtenues pour le robot mobile avant de trouver une solution au problème posé.

La nature du problème et de la méthode d'optimisation implique l'éventualité de se trouver bloqué dans des minima locaux différents du but. Par exemple, un minimum local peut correspondre à un plan optimal au sens de la fonction d'évaluation, qui nous amène à une collision avant d'arriver à la configuration finale (voir figure 1.3).

D'autre part, un problème majeur se pose dans la détermination d'une valeur minimale pour ℓ car on ne connaît pas le nombre de commandes à exécuter pour arriver à $\hat{q}_\bullet$.

Ces deux problèmes peuvent néanmoins être résolus dès lors que l'on s'intéresse aux trajectoires qui passent à plus de ε des $\mathcal{C}$-*obstacles*. On peut dans ce cas construire un arbre des plans qui permet d'explorer tout l'espace. Cet arbre est

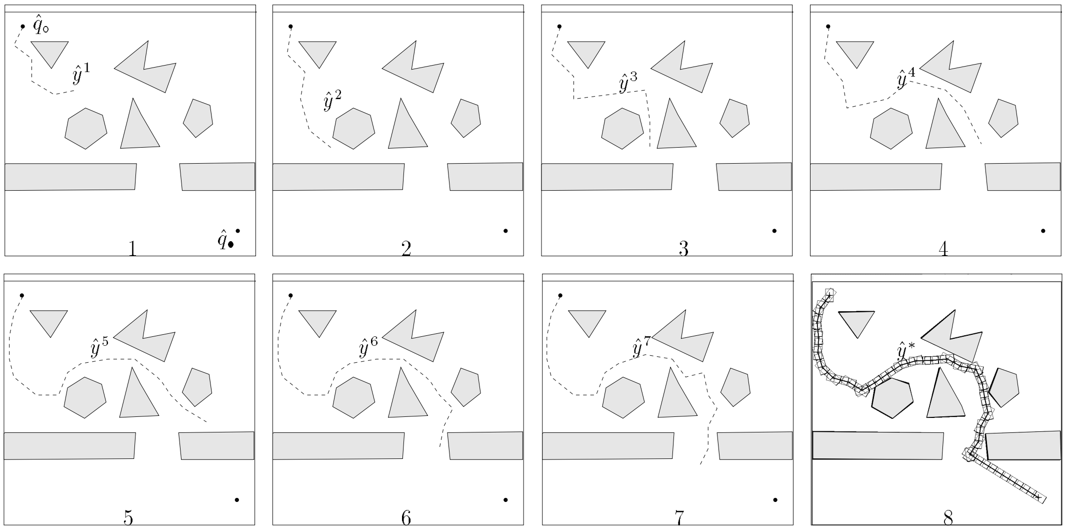


Fig. 1.2 - Séquence de solutions trouvées par un algorithme d'optimisation.

obtenu en posant des balises dans l'espace accessible du robot. Il permet d'une part de créer de nouveaux points de départ, et d'autre part de déterminer la valeur de ℓ qui permet d'aller au but. Le principe est de disposer des balises dans les régions inexplorées de l'espace des configurations. La figure 1.4 montre un exemple de la construction d'un tel arbre.

Nous montrerons par la suite que la stratégie de placement des balises construit implicitement une approximation de l'espace accessible depuis la configuration initiale.

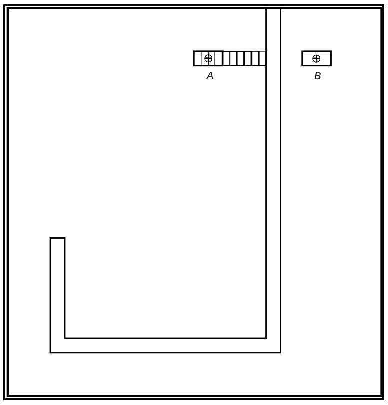


Fig. 1.3 - Un problème avec un minimum local.

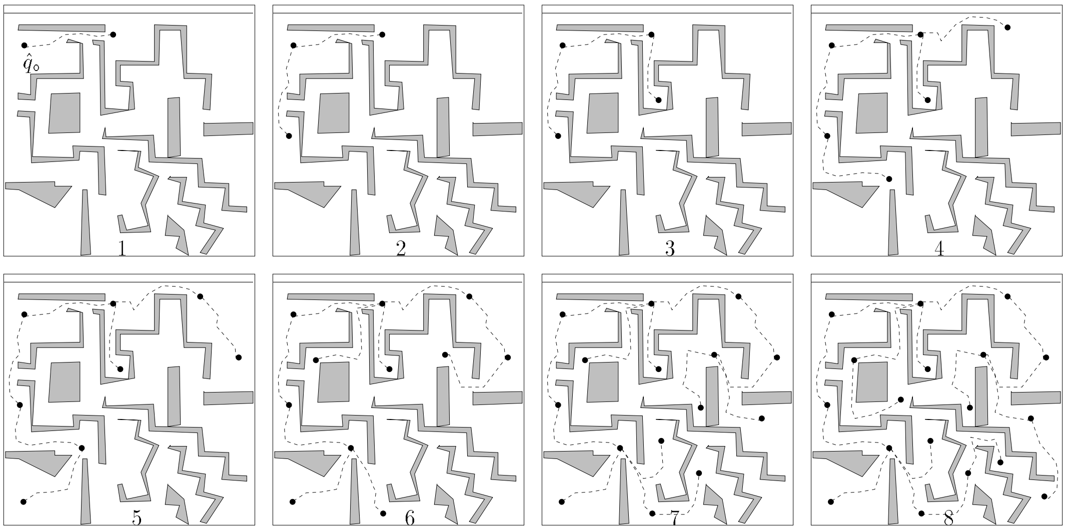


FIG. 1.4 - Construction de l'arbre des balises.

1.3 Résultats obtenus

Le principal résultat de cette thèse est l'élaboration d'une méthode de planification de trajectoires : L'algorithme Fil d'Ariane. Son originalité réside dans son adaptation automatique à la complexité du problème posé. La planification automatique de trajectoires est transformée en un problème d'optimisation d'une fonctionnelle de l'espace des plans dans $I\!R^+$. Cette optimisation permet d'explorer les positions accessibles à partir d'une configuration initiale. La méthode ne construit pas l'espace des configurations; par contre, à mesure que le temps passe, une approximation de plus en plus fine de l'espace accessible est construite. Cette approximation est faite par un premier algorithme appelé l'algorithme $EXPLORE$ qui garantit la complétude pour une résolution donnée de la méthode. D'autre part, un deuxième algorithme, l'algorithme $SEARCH$, permet d'accélérer la recherche d'un chemin et d'atteindre la configuration finale. En résumé, la méthode proposée permet de construire un planificateur complet pour une résolution donnée qui exploite d'une manière efficace l'espace des plans pour explorer l'espace des configurations.

Nous avons implanté deux planificateurs basés sur l'algorithme Fil d'Ariane : le premier est un planificateur de trajectoires pour un robot mobile holonome, et le deuxième, notre expérimentation principale, un planificateur de trajectoires pour un bras manipulateur à six degrés de liberté. Pour ce dernier, nous avons réalisé une implantation de notre algorithme sur une machine massivement parallèle et planifié les mouvements d'un bras à 6 degrés de liberté. Ce système est composé :

- d'un système de CAO robotique pour modéliser l'environnement 3D et les robots,
- d'un système de contrôle commande pour contrôler les robots,
- d'une machine massivement parallèle pour planifier les trajectoires.

Nous avons montré que l'utilisation d'une machine parallèle nous permet d'obtenir un planificateur de trajectoires rapide qui peut réagir aux changements de l'environnement. Pour démontrer cela, nous avons placé notre robot dans un environnement dynamique où un autre robot exécute des trajectoires aléatoires. Ces performances sont obtenues grâce à trois niveaux de parallélisation : le premier pour les algorithmes $SEARCH$ et $EXPLORE$, le deuxième pour la méthode d'optimisation et le dernier pour les calculs géométriques. La machine parallèle utilisée est composée de 128 Transputers.

Pour optimiser les fonctions des algorithmes $EXPLORE$ et $SEARCH$ nous avons utilisé les *algorithmes génétiques*. Ces algorithmes sont des méthodes d'optimisation stochastiques facilement parallélisables.

L'algorithme Fil d'Ariane a aussi été utilisé dans l'équipe robotique du LIFIA pour la planification de mouvements fins. Le planificateur proposé est capable de planifier des trajectoires en présence d'incertitude [22]. Les contacts du robot sont utilisés pour réduire l'incertitude de position et d'orientation. Deux types de trajectoires sont utilisés : des trajectoires sans contact et des trajectoires qui maintiennent le contact avec les obstacles. Les deux types de trajectoires sont utilisés par l'algorithme $EXPLORE$ pour engendrer de nouvelles balises et par l'algorithme $SEARCH$ pour atteindre le but. Une simulation en 2D a été développée.

Dans la suite de ce chapitre nous présenterons une introduction à l'algorithme Fil d'Ariane. Tout d'abord, nous donnerons le principe général de la méthode et introduirons les concepts de "pré-image" et de "post-image". Ensuite, nous décrirons plus en détail les algorithmes $SEARCH$ et $EXPLORE$. Finalement, nous présenterons le plan de ce document.

1.4 Introduction à l'algorithme Fil d'Ariane

1.4.1 Présentation de l'algorithme

Le rôle ultime d'un planificateur de trajectoires est de trouver un chemin dans l'espace des configurations depuis une position initiale jusqu'à une position finale.

Cependant un sous-but intéressant peut être d'essayer de collecter au passage des informations sur les positions de l'espace qui sont accessibles depuis la position initiale. L'algorithme Fil d'Ariane effectue simultanément les deux opérations grâce à deux sous-algorithmes **SEARCH** et **EXPLORE**.

L'algorithme EXPLORE collecte des informations sur l'espace accessible depuis la position initiale en posant des **balises** dans l'espace de recherche et en mémorisant le chemin entre ces balises et la position initiale. De façon à récolter une information aussi riche que possible sur l'espace accessible, EXPLORE essaye de placer ses balises aussi loin que possible des balises déjà placées. Comme on peut le montrer formellement, cette stratégie garantit de pouvoir placer une balise à une distance arbitrairement petite de tout point de l'espace des configurations accessible depuis la position initiale. A l'aide de cet algorithme on peut explorer tout l'espace accessible. Depuis chaque balise placée par EXPLORE, SEARCH utilise une méthode locale pour atteindre la position finale. Cette méthode garantit de trouver une solution si le but est totalement visible dans un voisinage de la position finale. Autrement dit, si une balise est placée dans ce voisinage, un chemin sera trouvé entre cette balise et la position finale et en conséquence entre la position initiale et la position finale. On peut donc montrer que l'algorithme Fil d'Ariane trouve toujours une solution s'il en existe une. Mis à part le voisinage du but, "SEARCH" définit implicitement une région de l'espace des configurations depuis laquelle on peut atteindre le but. On appelle cette région la "pré-image" du but. L'efficacité de l'algorithme Fil d'Ariane repose sur la grandeur de cette pré-image. Plus cette pré-image est grande plus la probabilité de placer rapidement une balise dans cette pré-image devient grande et en conséquence plus la solution peut être trouvée rapidement. En pratique, la pré-image du but occupe une vaste portion de l'espace des configurations et il n'est nécessaire de placer qu'un nombre limité de balises avant qu'une solution soit trouvée. Plus précisément, le temps de planification dépend de la facilité à atteindre la pré-image. Par exemple, si la pré-image du but contient la position initiale (ou première balise) le temps de calcul sera minimum. Par contre, s'il faut passer par un couloir de l'espace des configurations pour atteindre le but alors EXPLORE devra placer des balises avec une résolution égale à la dimension de l'entrée de ce couloir avant de pouvoir trouver une solution. Cet exemple montre que l'Algorithme Fil d'Ariane s'adapte naturellement à la véritable complexité du problème de planification posé, celle-ci ne dépendant pas forcément du nombre d'obstacles.

SEARCH et EXPLORE sont tous deux des algorithmes d'optimisation implantés sous la forme d'algorithmes génétiques parallèles. Il utilisent tous deux la notion de chemin Manhattan et les mêmes calculs géométriques de base que nous détaillerons dans les chapitres suivants.

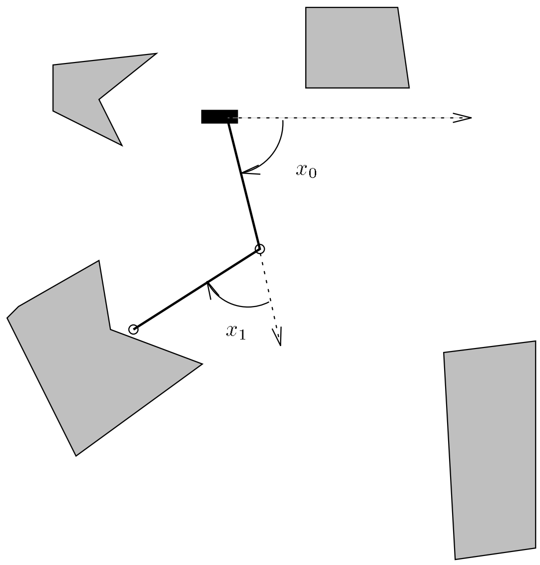


FIG. 1.5 - L'espace de travail d'un robot planaire à deux degrés de liberté.

L'algorithme SEARCH et la "pré-image"

La figure 1.6a représente l'espace des configurations du robot de la figure 1.5. Dans cet espace les *$\mathcal{C}$-obstacles* apparaissent en noir. Une fois encore, le calcul de ces *$\mathcal{C}$-obstacles* ne fait pas partie de notre méthode mais est utilisé ici pour la présenter. Soit $\mathcal{C}_{libre}$ le complémentaire des *$\mathcal{C}$-obstacles*; alors $\mathcal{C}_{libre_{\hat{q}_\bullet}}$ est le sous ensemble de $\mathcal{C}_{libre}$ connexe à la configuration $\hat{q}_\bullet$. Grâce à ces concepts et étant donnés un but $\hat{q}_\bullet$ et un planificateur $\mathcal{L}$, il est possible de définir une région de l'espace des configurations où le planificateur $\mathcal{L}$ réussira à produire une trajectoire valide pour atteindre la configuration $\hat{q}_\bullet$. Nous appelons cette région la "pré-image" de $\hat{q}_\bullet$ produit par $\mathcal{L}$ [41][46][47][23][53]. Si le planificateur $\mathcal{L}$ est un planificateur global, la "pré-image" produite est égale à la composantes connexe $\mathcal{C}_{libre_{\hat{q}_\bullet}}$ de l'espace libre auquel appartient $\hat{q}_\bullet$. Si $\mathcal{L}$ est un planificateur local, la "pré-image" est juste un sous-ensemble de $\mathcal{C}_{libre_{\hat{q}_\bullet}}$. Par exemple, dans la figure 1.6b nous avons représenté par la zone grise la pré-image de $\hat{q}_\bullet$ produite par un planificateur local. Bien que cette "pré-image" ne couvre pas tout l'espace $\mathcal{C}_{libre_{\hat{q}_\bullet}}$, elle peut en couvrir une grande partie.

Pour tout planificateur $\mathcal{L}$ et toute paire de configurations $\{\hat{q}_\circ, \hat{q}_\bullet\} \in \mathcal{C}_{libre}$ nous pouvons associer le prédicat $Chemin_{\mathcal{L}}(\hat{q}_\circ, \hat{q}_\bullet)$ qui est vrai si le planificateur est capable de trouver une trajectoire valide de la configuration initiale $\hat{q}_\circ$ à la configuration finale $\hat{q}_\bullet$. Ainsi, nous pouvons introduire la définition suivante :

Pour une configuration $\hat{q}_\bullet \in \mathcal{C}_{libre}$ et un planificateur $\mathcal{L}$, la *"pré-image" de $\hat{q}_\bullet$ produite par le planificateur $\mathcal{L}$* est définie comme :

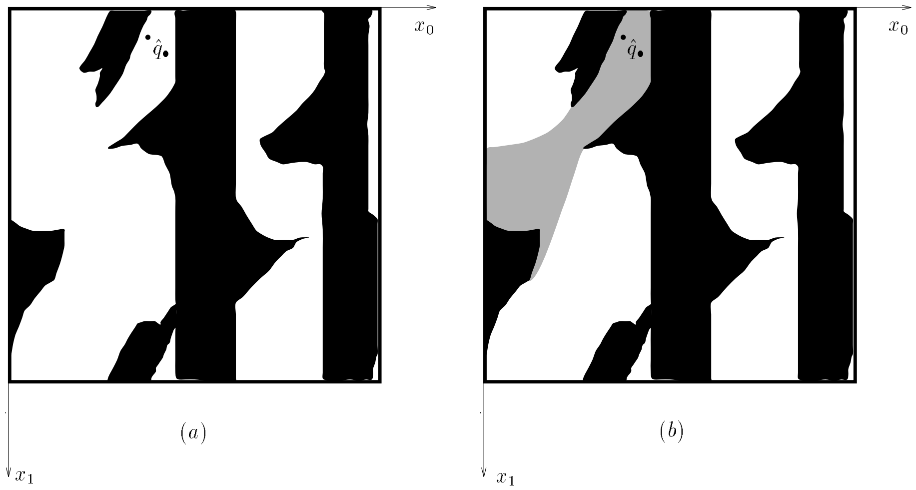


FIG. 1.6 - La “pré-image” de $\hat{q}_\bullet$ produite par le planificateur $\mathcal{L}$.

$$\mathcal{P}_{\mathcal{L}}(\hat{q}_\bullet) = \{\hat{q} \in \mathcal{C}_{libre} | Chemin_{\mathcal{L}}(\hat{q}, \hat{q}_\bullet) = vrai\}$$

Evidemment un planificateur global ne peut être construit que s'il est possible de planifier un chemin valide de la position initiale à la position finale : tel est le rôle de l'algorithme EXPLORE.

L'algorithme EXPLORE et la “post-image”

L'algorithme EXPLORE produit un ensemble de balises $EL = \{L_1, L_2, \dots, L_t\}$ pour lesquelles un chemin est connu à partir de la configuration initiale. Le but de l'algorithme EXPLORE est de placer une balise appartenant à la “pré-image” de la configuration finale produite par SEARCH. Comme nous n'avons pas une connaissance a priori ni de la taille ni du placement de la “pré-image”, la meilleure stratégie est de placer des balises d'une manière équitablement répartie dans l'espace libre. Ce placement est effectué grâce à un algorithme générateur de trajectoires A_G. Ainsi, au fur et à mesure que les balises sont placées, les chemins connectant ces balises sont sauvegardés sous la forme d'un arbre (voir figure 1.7).

Etant donné un ensemble de balises placées dans l'espace des configurations $EL = \{L_1, L_2, \dots, L_t\} \subset \mathcal{C}_{libre}$ et le planificateur $\mathcal{L}$, nous définissons la *“post-image”* de EL produite par $\mathcal{L}$ comme :

$$\mathcal{P}st_{\mathcal{L}}(EL) = \{\hat{q} \in \mathcal{C}_{libre} | Chemin_{\mathcal{L}}(L_i, \hat{q}) = vrai, L_i \in EL\}$$

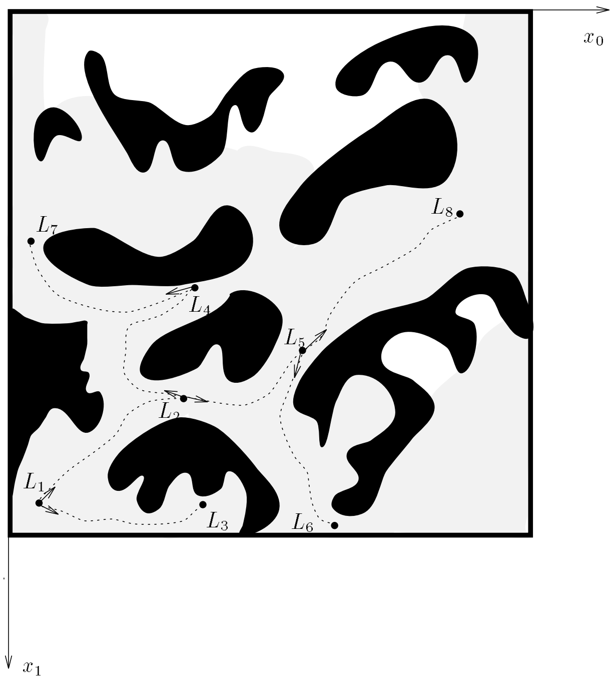


FIG. 1.7 - La "post-image" de EL produite par le planificateur $\mathcal{L}$.

Il est possible de montrer qu'en suivant cette procédure la composante connexe $\mathcal{C}_{libre_{\hat{q}_0}}$ peut être approximée par un ensemble EL de balises avec une résolution arbitraire. Dans la figure 1.7 la "post-image" de l'ensemble $EL = \{L_1, L_2, \ldots, L_8\}$ est représentée par la zone grise.

1.4.2 L'algorithme SEARCH

Le principe de l'algorithme SEARCH est de déterminer si une configuration $\hat{q}_\bullet$ peut être atteinte à partir d'une configuration donnée $\hat{q} \in \mathcal{C}_{libre}$. Ainsi, l'espace de recherche de SEARCH est l'ensemble des trajectoires partant de la configuration $\hat{q}$ et produites par l'algorithme générateur de trajectoires valides A_G.

Etant donnée une configuration $\hat{q} \in X \subset I\!\!R^n$, l'ensemble $\omega(X; \hat{q})$ est défini comme l'ensemble des chemins en X partant de la configuration $\hat{q}$.

Nous notons par $\hat{\varphi}$ tout chemin appartenant à $\omega(X;\hat{q})$ et par $E(\hat{\varphi})$ l'extrémité de ce chemin. Ainsi pour une configuration $\hat{q}_\bullet$ il est possible de définir la fonction $F:\omega(\mathcal{C}_{libre_{\hat{q}}};\hat{q}) \rightarrow I\!R^+$ comme :

$$F(\hat{\varphi},\hat{q}_\bullet) = ||E(\hat{\varphi}) - \hat{q}_\bullet||$$

Comme le générateur A_G utilisé engendre uniquement un ensemble $M_G \subset \omega(X;\hat{q})$ de trajectoires, l'algorithme SEARCH peut être exprimé comme un problème de minimisation :

$$SEARCH(\hat{q},\hat{q}_\bullet) = \min\{F(\hat{\varphi},\hat{q}_\bullet)|\hat{\varphi} \in M_G \subset \omega(X;\hat{q})\}$$

De ce fait, s'il existe une trajectoire en M_G de $\hat{q}$ à $\hat{q}_\bullet$, alors $SEARCH(\hat{q},\hat{q}_\bullet) = 0$.

1.4.3 L'algorithme EXPLORE

Un des principaux problèmes des méthodes de planification de trajectoires est l'absence d'un compromis entre réaliser une exploration de l'espace libre et chercher le but. Par exemple, pour un algorithme utilisant une approche de type potentiel, il est évident que, quelle que soit la fonction potentielle employée, la méthode a un comportement "élitiste". Certaines trajectoires comme celles qui s'approchent tout de suite du but sont privilégiées; en revanche, les trajectoires qui amènent le robot loin du but ne sont pas toujours explorées. Cet "élitisme" entraine l'exploration répétitive de certaines zones de l'espace des configurations et l'on peut, par exemple, tomber plusieurs fois dans le même minimum local. L'algorithme EXPLORE est indépendant du but; il réalise une exploration de l'espace libre d'une manière équitable. Le compromis de l'algorithme EXPLORE est de favoriser les régions de l'espace des configurations les moins explorées. Pour cela, EXPLORE commence la recherche avec des régions implicites très grandes. A mesure que le temps passe il réduit la taille des régions pour augmenter le niveau d'exploration.

Nous pouvons imaginer l'algorithme EXPLORE comme un robot en train de placer des balises dans l'espace libre. L'algorithme commence avec la configuration initiale $\hat{q}_\circ$ comme première balise, c'est-à-dire $EL = \{L_1 = \hat{q}_\circ\}$. A chaque fois qu'une nouvelle balise sera placée EXPLORE essaie de trouver la région de l'espace libre la moins explorée pour placer la nouvelle balise, ce qui revient à trouver la configuration la plus éloignée des balises précédemment placées. Cette nouvelle balise est attachée par un "fil d'Ariane" à une des balises précédemment placées. Il est possible de définir la distance d'un point $y \in I\!R^n$ à un ensemble $X \subset I\!R^n$ par :

$$d(y, X) : \min_{x \in X} ||y - x||$$

Ainsi, pour toute configuration $\hat{q} \in \mathcal{C}_{libre_{\hat{q}_0}}$, le degré d'exploration est inversement proportionnel à $d(\hat{q}, EL)$. Notons par PL_{EL} l'ensemble des trajectoires engendrées par l'algorithme A_G ayant comme point de départ une des balises de EL. L'objectif est de trouver des trajectoires qui nous amènent aux régions les moins explorées.

Pour la première phase de l'algorithme nous cherchons alors :

$$\hat{\varphi}_1 : \max_{\hat{\varphi} \in PL_{EL}} d(E(\hat{\varphi}), L_1)$$

et nous faisons $L_2 = E(\hat{\varphi}_1)$ ainsi, $EL = EL \cup \{L_2\}$. De manière similaire $\hat{\varphi}_2$ est définie comme :

$$\hat{\varphi}_2 : \max_{\hat{\varphi} \in PL_{EL}} d(E(\hat{\varphi}), EL)$$

Plus particulièrement, si nous avons n balises, nous pouvons trouver la balise $(n+1)$ en maximisant l'expression :

$$\hat{\varphi}_n : \max_{\hat{\varphi} \in PL_{EL}} d(E(\hat{\varphi}), EL)$$

et $L_{n+1} = E(\hat{\varphi}_n)$. Considérons la fonction :

$$EXPLORE(n) = \max_{\hat{\varphi} \in PL_{EL}} d(E(\hat{\varphi}), EL)$$

alors si l'espace $\mathcal{C}_{libre_{\hat{q}_0}}$ est borné nous avons:

$$\lim_{n \to \infty} EXPLORE(n) = 0$$

autrement dit :

$$\forall \varepsilon \ \ \exists k \ \ : \ \ \forall K > k \ \ EXPLORE(K) < \varepsilon$$

Ceci revient à dire que pour toute configuration $\hat{q} \in \mathcal{C}_{libre_{\hat{q}_0}}$ il existe une balise L_i telle que $||\hat{q} - L_i|| < \varepsilon$. Nous appelons cette propriété *l'approximation à epsilon près* de $\mathcal{C}_{libre_{\hat{q}_0}}$. Cette dernière propriété a une conséquence très importante pour la planification d'un chemin dans un espace continu : si nous pouvons trouver ε tel qu'il existe une fonction qui résout le problème de planification de trajectoires pour tout sphère de rayon ε (la fonction SEARCH), alors en la combinant avec l'algorithme EXPLORE nous obtenons une méthode pour planifier des chemins entre deux points arbitraires de cet espace.

Nous pouvons à présent définir l'algorithme Fil d'Ariane de la manière suivante :

Début : $L_1 \leftarrow \hat{q}_\circ, EL \leftarrow \{L_1 = \hat{q}_0\}, i \leftarrow 1$

1. Exécution de SEARCH : Si $SEARCH(L_i, \hat{q}_f) = 0$ alors un chemin a été trouvé.
2. Exécution d'EXPLORE : Placer une nouvelle balise L_{i+1} avec la fonction $EXPLORE(i)$, $EL \leftarrow EL \cup \{L_{i+1}\}$, $i \leftarrow i+1$ *goto 1*

L'algorithme Fil d'Ariane a trois caractéristiques très importantes :

- il adapte la résolution de recherche à la complexité du problème.
- Il est complet pour une résolution donnée[1], c'est-à-dire que si un chemin existe pour cette résolution il sera trouvé par l'algorithme.
- Il ne nécessite pas le calcul explicite de l'espace des configurations.

Dans les chapitres suivants nous introduirons un type particulier de trajectoires, les trajectoires Manhattan. Notre approche est basée sur l'utilisation de ce type particulier de trajectoire pour engendrer des trajectoires paramétrables.

1.5 Organisation du document

Le présent mémoire est constitué par :

1. **une partie théorique** exposant les bases mathématiques de l'algorithme Fil d'Ariane,
2. **une partie expérimentale** présentant deux planificateurs de trajectoires basés sur cet algorithme :

[1] "resolution-complet"

(a) un planificateur de trajectoires pour un robot mobile holonome

(b) un planificateur de trajectoires parallèle pour un bras manipulateur à six degrés de liberté.

Avant de situer notre travail parmi les travaux précédemment effectués nous explicitons l'organisation du document.

1.5.1 Partie I (Le Fil d'Ariane : un algorithme général de planification)

Si dans la partie précédente nous avons présenté l'algorithme Fil d'Ariane en introduisant d'abord l'algorithme $SEARCH$, puis l'algorithme $EXPLORE$, nous l'avons fait pour des raisons historiques et pédagogiques. Dans cette partie, nous présentons d'abord l'algorithme $EXPLORE$ comme le cœur de l'algorithme Fil d'Ariane. Trois chapitres sont présentés comme suit :

Dans le **chapitre 2** nous définissons tout d'abord les concepts formels permettant d'exposer les bases mathématiques de l'algorithme $EXPLORE$: espace des configurations, chemin dans $\mathbb{R}^n$, ensemble chemin connecté, distance d'Hausdorff, billage et pavage d'un espace.

Dans le **chapitre 3**, nous définissons une première méthode d'exploration : $EXPLORE_\infty$. Puis, nous introduisons les notions d'espace libre à ε-près et de chemin Manhattan. Enfin, en utilisant ces deux nouveaux concepts, nous présentons l'algorithme $EXPLORE$, un algorithme dérivé d'$EXPLORE_\infty$. Nous montrons dans ce chapitre que l'algorithme $EXPLORE$ est un algorithme complet pour une résolution donnée et qu'il peut être exprimé comme un ensemble de problèmes d'optimisation sur $\mathbb{R}^{\ell * n}$.

Dans le **chapitre 4**, deux améliorations de l'algorithme $EXPLORE$ sont apportées : l'algorithme $SEARCH$ et le rebondissement des trajectoires. Nous montrons que l'algorithme $SEARCH$ peut être aussi exprimé comme un problème d'optimisation sur $\mathbb{R}^{\ell * n}$. Enfin, l'algorithme Fil d'Ariane est défini comme la combinaison des deux problèmes d'optimisation : $EXPLORE$ et $SEARCH$.

1.5.2 Partie II (Application au problème de planification en robotique)

Cette partie est composée de cinq chapitres présentant : les algorithmes génétiques, l'implantation des deux planificateurs, la parallélisation du système pour le bras manipulateur et les conclusions.

Dans le **chapitre 5**, nous présentons tout d'abord les algorithmes génétiques. Cette technique a été utilisée pour optimiser les fonctions de $SEARCH$ et $EXPLORE$.

Pour introduire les algorithmes génétiques, nous donnons un exemple d'application : le problème de la fonction cinématique inverse. Puis, nous décrivons les algorithmes génétiques parallèles. Une évaluation des performances de ce type d'algorithme est aussi présentée. Finalement, nous explicitons les raisons qui nous ont encouragés à utiliser des algorithmes génétiques comme technique d'optimisation.

Dans le **chapitre 6**, un planificateur de trajectoires pour un robot mobile holonome est présenté. L'objectif visé est de montrer la rapidité du planificateur proposé et la possibilité de construire un planificateur global réactif. Une discussion sur le comportement de l'algorithme $EXPLORE$ est menée. Dans cette discussion, on aborde le sujet de l'influence de la technique d'optimisation et l'influence des chemins utilisés. On montre également comment il est possible d'obtenir des informations intéressantes à-propos de l'espace des configurations lorsqu'on exécute l'algorithme $EXPLORE$.

Dans le **chapitre 7**, nous présentons un planificateur de trajectoires pour un bras manipulateur à six degrés de liberté. En premier lieu, nous décrivons le fonctionnement du système et son utilisation. Puis, nous présentons le système de modélisation du monde réel. Ce modèle est basé sur une représentation hiérarchique à trois niveaux. On montre que chacun des niveaux a une fonction spécifique. On expose ensuite la méthode de calcul de débattements permettant d'obtenir des trajectoires valides. Trois outils servant à ce calcul sont décrits : les types de contact, la mise à jour du robot et les transformations homogènes. Nous décrivons ensuite le processus de décodage d'une trajectoire. Nous terminons ce chapitre par une brève discussion sur l'expérimentation.

Le **chapitre 8**, décrit l'implantation parallèle du planificateur de trajectoires pour le bras manipulateur. Trois niveaux de parallélisation ont été implantés. Le premier niveau comprend l'exécution de $SEARCH$ et $EXPLORE$, le deuxième les algorithmes génétiques et le troisième la fonction d'évaluation. Nous décrivons tout d'abord la logique de ces niveaux ainsi que les processus qui les composent. Ensuite, nous présentons le placement des différents processus sur les processeurs de la machine parallèle.

Enfin, les perspectives de cette thèse sont exposées au **chapitre 9**. Premièrement, nous discutons les possibles directions de recherche pouvant faire suite à notre travail. Puis, nous rappelons les principaux résultats théoriques et expérimentaux obtenus. Finalement, une discussion à-propos de la méthode proposée est abordée.

1.6 Cadre de l'étude

Nous rappelons que le problème de la planification automatique de trajectoires consiste à établir des méthodes permettant au robot de "décider", d'une manière autonome, ses mouvements pour atteindre un but donné, c'est-à-dire de trouver une séquence de mouvements qui permet de passer d'un état initial à un état final. Cette séquence de mouvements doit être libre de toute collision avec les obstacles, de plus, elle doit respecter certaines contraintes liées à la cinématique du robot, à l'environnement, etc... et certains critères fixés par l'utilisateur : optimalité en temps, en énergie etc.... Le problème de la planification de trajectoire n'est donc pas monolithique et peut revêtir de très nombreuses formes. Hwang et Ahuja [36] proposent une classification en répartissant les différents problèmes sur une grille à deux entrées l'une correspondant aux types d'environnement traités et l'autre aux types de robot considérés. Le lecteur intéressé peut trouver une synthèse de ces travaux dans l'annexe A.

1.7 Travaux proches de notre méthode

Bien que plusieurs des méthodes présentées dans cette annexe garantissent la résolution du problème, leurs implantations pour plusieurs degrés de liberté (plus de 3) deviennent la plupart du temps impossible. Cette limitation est directement reliée à la complexité des méthodes proposées. De plus, si nous voulons considérer des espaces dynamiques, la complexité devient inconcevable.

De nouvelles techniques de planification de trajectoires continuent à être développées. Ces nouvelles techniques sont, en générale, des approches mixtes. Leur objectif est la construction incrémentale de l'espace des configurations par des éléments représentatifs de régions ou portions de l'espace; ces portions n'étant pas nécessairement explorées [36]. D'un autre côté l'utilisation de systèmes parallèles est de plus en plus étudiée et l'emploi de ceux-ci semble nécessaire si l'on veut planifier des trajectoires en temps-réel [16][50][71]. Bien que de très nombreuses solutions aient été proposées, le paragraphe suivant présente uniquement les travaux qui s'approchent le plus de notre travail.

1.7.1 La méthode stochastique

Il s'agit d'une technique qui combine l'approche potentielle avec une recherche aléatoire [40][10][11][60] [61] [77]. Pour un espace des configurations $\mathcal{C}$ de dimension n l'axe de coordonnées est noté par $x_0, x_1, \ldots, x_{n-1}$. Cet espace est divisé en cellules où un déplacement dans la direction de l'axe x_i est représenté par Δ_i.

Soit U la fonction potentielle, $\hat{q}$ un point $(x_0, \ldots x_{n-1})$ dans l'espace et $\hat{q}_\bullet$ la configuration finale alors $\forall\ \hat{q} \in \mathcal{C}$, $U(\hat{q}) \geq 0$. Si $U(\hat{q}) = 0$ alors $\hat{q} = \hat{q}_\bullet$.

Le planificateur construit incrémentalement un graphe G tel que les sommets du graphe soient des minima locaux de la fonction potentielle U. Deux minima locaux sont connectés si le planificateur a construit un chemin entre eux. L'algorithme commence la recherche avec la position initiale $\hat{q}_\circ$. A partir de cette position, il exécute un mouvement " meilleur-d'abord" c'est-à-dire l'algorithme passe de la position $\hat{q}_\circ$ à $\hat{q}_1$ tel que $\hat{q}_1$ est un voisin de $\hat{q}_\circ$. La position $\hat{q}_1$ est égale à $\hat{q}_\circ + \Delta_i$ et $U(\hat{q}_\circ) \geq U(\hat{q}_1)$. Le planificateur fait de même pour $\hat{q}_2, \ldots \hat{q}_{loc}$ où $\hat{q}_{loc}$ est un minimum local. Si $U(\hat{q}_{loc}) = 0$ alors l'algorithme a trouvé une solution. Sinon si $U(\hat{q}_{loc}) > 0$ le planificateur essaie d'échapper au minimum local par l'exécution d'un ensemble de mouvements aléatoires à partir de $\hat{q}_{loc}$. Chaque mouvement aléatoire est suivi d'une recherche meilleur-d'abord qui attend un minimum local. Si ce minimum $\hat{q}'_{loc}$ est différent de $\hat{q}_{loc}$, il est remplacé par $\hat{q}'_{loc}$ "le successeur" de $\hat{q}_{loc}$ dans le graphe G. A ce moment-là, les deux minima adjacents sont connectés par le chemin qui a été trouvé par la recherche composée d'un mouvement aléatoire et d'une autre mouvement meilleur-d'abord. Le graphe G est augmenté jusqu'à trouver la configuration finale. Si la recherche se finit avec succès, le chemin construit est transformé en un chemin régulier. Cette transformation réalise une optimisation de la trajectoire engendrée. Les expériences faites avec cette méthode montrent qu'elle est très efficace. Cependant, le temps de recherche augmente d'une manière exponentielle à mesure que le nombre de degrés de liberté augmente.

Etant donné que l'algorithme utilise une procédure aléatoire pour la construction du graphe des minima locaux, on ne peut pas garantir qu'une solution sera trouvée même si elle existe. Par contre, il est possible de prouver qu'au fur et à mesure que l'on engendre des déplacements aléatoires à partir des minima locaux, la probabilité d'arriver au but converge vers 1. Entre autre, l'algorithme engendre des chemins différents s'il est lancé plusieurs fois sur un même problème. Un autre problème de cette méthode est le nombre de voisins à considérer lorsque le nombre de degrés de liberté augmente. Le nombre de voisins d'un point $\hat{q}_i \in I\!R^n$ est $3^n - 1$. Pour $n = 6$ le point $\hat{q}_i$ a 728 voisins possibles. Une solution à ce problème est de sélectionner uniquement quelques voisins par un tirage aléatoire [10].

1.7.2 La construction incrémentale

Cette méthode est une des plus récentes et consiste à construire incrémentalement un squelette ou une rétraction représentant l'espace libre à partir de la configuration initiale ou finale, c'est-à-dire en réalisant une recherche unidirectionnelle ou bidirectionnelle. Le but est la construction d'un squelette homotopique à tous les chemins de l'espace libre. Si nous cherchons un chemin quelconque d'une position initiale à une autre finale la construction de ce squelette est arrêtée lorsqu'un chemin est trouvé.

Ce type d'approche est normalement combiné avec un algorithme local $\mathcal{L}$ qui produit (à partir des positions déjà calculées) de nouvelles positions libres ou bien vérifie si la position finale peut être atteinte [15][55][18].

Un premier exemple de cette méthode est la stratégie de recherche appelée SANDROS [18]. L'algorithme est composé de deux planificateurs : un planificateur global $\mathcal{G}$ et un planificateur local $\mathcal{L}$. Le planificateur $\mathcal{G}$ engendre une séquence de sous-buts pour guider le robot, et le planificateur $\mathcal{L}$ vérifie si ces sous-buts sont accessibles à partir des positions déjà atteintes. Cette stratégie a été utilisée pour produire des trajectoires sans collision pour un bras manipulateur à six degrés de liberté. L'algorithme présente deux caractéristiques principales : il définit les sous-buts d'une manière hiérarchique avec plusieurs niveaux de résolution et il réalise une recherche bidirectionnelle. Les sous-buts de cet algorithme sont des sous-espaces rectangulaires de dimension $0, 1, \ldots, n$ ou n est le nombre de degrés de liberté. Ainsi, si on note par des majuscules les coordonnées non unifiées d'un point $x \in \mathcal{C} \subset I\!\!R^n$ nous pouvons représenter des sous-espaces de l'espace des configurations. Par exemple, pour un robot à six degrés de liberté, le sous espace de dimension six $(X_0, X_1, X_2, X_3, X_4, X_5)$ représente tout l'espace $I\!\!R^6$, le sous-espace $(X_0, x_1, x_2, x_3, x_4, x_5)$ représente un sous espace de dimension 1 et $(x_0, x_1, x_2, x_3, x_4, x_5)$ un point bien précis. L'algorithme consiste à construire un graphe G dans lequel les sommets représentent des points ou des sous-espaces de ce type. Ces derniers ne sont pas nécessairement explorés. Il y a alors deux types de sommets; les sommets "parvenus"et les sommets "non-parvenus". Un sommet n_i est parvenu à partir d'un point $\hat{q}$ si $\mathcal{L}$ est capable de trouver une chemin libre de $\hat{q}$ à n_i. Tout d'abord le graphe est constitué de deux sommets, celui de la configuration initiale $\hat{q}_\circ$ et celui de la configuration finale $\hat{q}_\bullet$. Le planificateur $\mathcal{L}$ est lancé pour trouver un chemin de $\hat{q}_\circ$ à $\hat{q}_\bullet$, s'il n'y a pas de solution l'espace (initialement de dimension n) est "éclaté" en sous-espaces de dimension $n-1$, c'est-à-dire en construisant des nouveaux sommets du graphe. La recherche meilleur-d'abord est utilisée pour explorer le graphe G et vérifier que les nouveaux sommets sont accessibles. Tant qu'il n'y a pas un chemin du sommet de $\hat{q}_\circ$ au sommet de $\hat{q}_\bullet$ avec tous ses sommets en état "parvenu" on continue la recherche en G. Dès que le graphe est complètement exploré et si une solution n'a pas été trouvée les sommets de dimension supérieure à 0 sont "éclatés" en construisant de nouveaux sommets de dimension inférieure. Ce processus est répété jusqu'a ce qu'il trouve une solution ou qu'il ne soit pas possible "d'éclater" de nouveaux sommets.

Une autre méthode similaire est celle décrite en [15]. La méthode construit un squelette de l'espace libre grâce aux chemins libres créés par une fonction potentielle. Le but est de connecter des tranches de l'espace libre par des chemins le long de cet espace.

La principale ressemblance des méthodes précédemment décrites avec la nôtre est l'utilisation d'une stratégie pour définir des nouvelles configurations de départ; et, l'utilisation d'une approche locale pour chercher à atteindre ces configurations. La principale différence est que dans notre cas nous utilisons l'espace des plans et non pas l'espace des configurations pour atteindre les nouvelles configurations.

Partie I

Le Fil d'Ariane : Un algorithme de planification général

Introduction

Le but de cette partie est de décrire formellement un algorithme de planification général à l'aide de deux problèmes d'optimisation de $I\!R^n$ dans R^+. La démarche que nous avons suivie est la suivante :

Nous rappelons tout d'abord les définitions formelles des concepts fondamentaux utilisés : espace des configurations, chemin dans $I\!R^n$, ensemble "chemin connecté", distance de Hausdorff, billage et pavage d'un espace.

Cet ensemble de définitions nous permettra d'introduire une première méthode mathématique $EXPLORE_\infty$ qui permet de construire en un nombre fini d'étapes un ensemble discret de points qui approche l'espace libre arbitrairement près au sens de la distance de Hausdorff. Le résultat sur la convergence de cette méthode sera utilisé plus tard pour montrer la convergence d'un algorithme dérivé de la première méthode : $EXPLORE$.

Nous introduisons ensuite la notion de *Chemin Manhattan* et la notion d'*espace libre à ε-près* qui représente naturellement les portions de l'espace des configurations qui se trouvent à plus de ε des obstacles. Si l'on s'intéresse à des trajectoires entièrement contenues dans cet espace on peut, en utilisant les chemins Manhattan, obtenir une méthode de construction de l'espace atteignable basée sur la maximisation d'une fonctionnelle de $I\!R^n$ dans $I\!R^+$. Nous montrerons la convergence de cette méthode - $EXPLORE$ - c'est-à-dire que nous montrerons qu'on peut approcher l'espace atteignable avec une résolution arbitraire en un nombre fini d'itérations.

Le rôle premier de $EXPLORE$ est, bien entendu, de fournir une représentation approchée de l'espace atteignable qui peut être utilisée en planification de trajectoire. Cependant son intérêt majeur réside dans la manière dont cette représentation est calculée. En effet, cette méthode fournit une représentation de l'espace qui s'adapte naturellement à la complexité du problème de planification posé.

A ce stade, nous pourrions disposer d'un premier planificateur de trajectoire. En effet, en supposant que le but soit dans l'espace libre à ε-près et qu'il soit atteignable, alors la boule de rayon ε autour du but est libre de tout obstacle et contient une balise de l'ensemble construit par $EXPLORE$. Il nous suffit alors de construire la trajectoire qui va de l'origine à ce point, ce qui est possible par définition, et de joindre "en ligne droite" le but depuis ce point. En quelque sorte, la boule de rayon ε représente une pré-image minimale de la position but.

L'objet de l'algorithme $SEARCH$ est de fournir une méthode permettant d'agrandir cette pré-image et donc d'augmenter l'efficacité d' $EXPLORE$. L'intérêt de $SEARCH$ par rapport à d'autres méthodes de planification locales qui pourraient jouer un rôle similaire d'agrandissement, est d'utiliser les mêmes outils et concepts que ceux employés dans $EXPLORE$. En particulier $SEARCH$ s'exprime comme l'optimisation d'une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^+$. $SEARCH$ constitue donc un complément générique d'$EXPLORE$.

Nous montrerons comment on peut grandement améliorer l'efficacité de la recherche en introduisant la notion de rebondissement contre les *C-obstacles*. Finalement, nous définirons l'algorithme Fil d'Ariane comme la combinaison des algorithmes $EXPLORE$ et $SEARCH$.

Chapitre 2

Définitions

2.1 L'espace des configurations

On note $\mathcal{W}$ l'espace physique dans lequel évolue le robot considéré. En général on peut déterminer la position de tous les points du robot dans cet espace par un nombre restreint de paramètres $(x_1, x_2, \ldots, x_n)$. Le choix de ces paramètres définit un espace appelé *espace des configurations* [49][41] noté $\mathcal{C}$. Par définition, la position du robot dans $\mathcal{W}$ est complètement définie par la donnée d'un point dans $\mathcal{C}$. En général, des contraintes d'ordre mécanique ou physique empêchent le robot de se trouver dans certaines régions de $\mathcal{W}$. Ces régions définissent implicitement des régions interdites de $\mathcal{C}$: on nomme ces régions les $\mathcal{C}$-*obstacles*. Par exemple, pour un robot manipulateur, des contraintes mécaniques sont imposées par les butées de chaque degré de liberté. Ainsi, pour ce type de robot, l'espace des configurations $\mathcal{C}$ est borné. Les autres contraintes, celles d'ordre physique, sont imposées par les obstacles $\mathcal{B}_{i=0,\ldots k} \subset \mathcal{W}$ placés dans l'espace accessible au robot (voir figure 2.1). Evidemment, le robot et les obstacles ne peuvent pas occuper simultanément le même sous-espace de $\mathcal{W}$. Ainsi, toutes les configurations positionnant le robot dans l'espace occupé par un obstacle appartiennent, dans l'espace des configurations, aux $\mathcal{C}$-*obstacles*.

***Définition 2.1** (Emplacement d'un robot)*

*Soit $\mathcal{A}$ un robot, $\mathcal{C}$ l'espace des configurations de $\mathcal{A}$, $\hat{q} \in \mathcal{C}$ une configuration du robot et $\mathcal{W}$ l'espace de travail de $\mathcal{A}$. L'*emplacement de $\mathcal{A}$ dans $\mathcal{W}$ donné par $\hat{q}$ *est défini comme le sous-espace de $\mathcal{W}$ occupé par $\mathcal{A}$ étant données les valeurs des paramètres de q. Il est noté $\mathcal{A}(\hat{q})$.*

Avec la définition précédente, nous pouvons maintenant définir les *$\mathcal{C}$-obstacles* engendrés par l'ensemble "des obstacles physiques" $\mathcal{B}_{i=0,...k}$ placés dans $\mathcal{W}$. Chacun des obstacles physiques en $\mathcal{W}$ est transformé dans l'espace des configurations $\mathcal{C}$ de $\mathcal{A}$ en un ensemble de régions :

$$\mathcal{CB}_i = \{\hat{q} \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{B}_i \neq \emptyset\}$$

où $\mathcal{CB}_i$ peut être un ensemble vide ($\forall \hat{q} \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{B}_i = \emptyset$). De cette manière, l'union de ces régions définit l'ensemble de $\mathcal{CB} \subset \mathcal{C}$ des configurations interdites au robot.

***Définition 2.2** ($\mathcal{C}$-obstacles)*

L'ensemble des $\mathcal{C}$-obstacles noté $\mathcal{CB}$ est défini comme l'union des transformations de tous les obstacles du monde physique dans l'espace des configurations, c'est-à-dire :

$$\mathcal{CB} = \bigcup_{i=1}^{k} \mathcal{CB}_i$$

Par exemple, la figure 2.1 montre un bras manipulateur planaire en deux positions différentes et placé parmi quatre obstacles $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$. Deux paramètres angulaires x_0 et x_1 sont utilisés pour définir une configuration de ce type de robot. Ainsi, l'espace des configurations $\mathcal{C}$ est égal à $[0, 2\pi[\times[0, 2\pi[\subset I\!\!R^2$ (voir figure 2.2). Dans cet espace, chacune des deux positions du robot est représentée par un point et l'ensemble des positions interdites correspondant aux obstacles par les régions grisées. On peut remarquer que contrairement aux objets $\mathcal{B}_0, \mathcal{B}_2, \mathcal{B}_3$, la transformation de $\mathcal{B}_1$ sur $\mathcal{C}$ est un ensemble vide. En effet, aucune position de $\mathcal{A}$ ne conduit à une collision avec cet obstacle. D'autre part, on remarque que les transformations de $\mathcal{B}_2$ et $\mathcal{B}_3$ dans l'espace des configurations ne sont pas disjointes. Ainsi, la transformation de $\mathcal{CB}_2 \cap \mathcal{CB}_3$ dans l'espace $\mathcal{W}$ correspond aux emplacements pour lesquels le robot entre en collision à la fois avec les obstacles $\mathcal{B}_2$ et $\mathcal{B}_3$.

Le complémentaire de $\mathcal{CB}$ définit un sous-espace de $\mathcal{C}$ où le robot se trouve libre de collision. Cet espace est appelé *l'espace libre* :

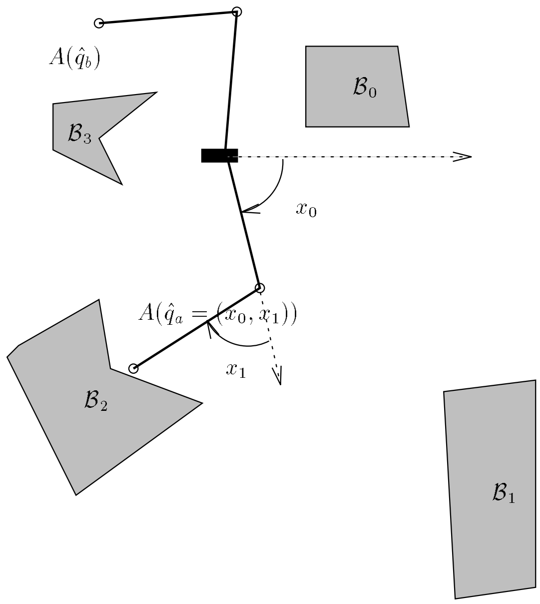


FIG. 2.1 - L'espace de travail d'un robot planaire à deux degrés de liberté.

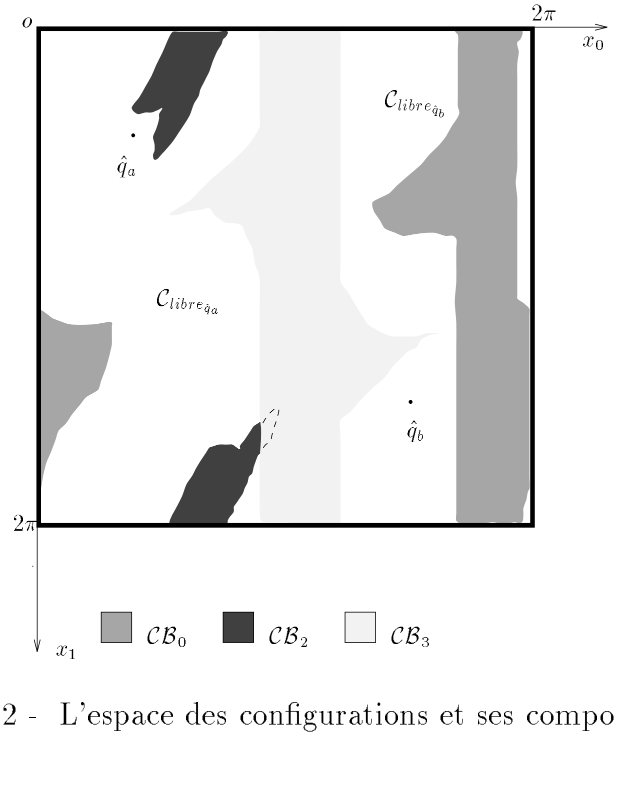


FIG. 2.2 - L'espace des configurations et ses composantes.

Définition 2.3 (Espace libre)
*L'*espace libre*, noté $\mathcal{C}_{libre}$, d'un robot $\mathcal{A}$ est défini comme l'ensemble des configurations libres de collision de $\mathcal{A}$:*

$$\mathcal{C}_{libre} = \mathcal{C} \backslash \mathcal{CB} = \{\hat{q} \in \mathcal{C} | \mathcal{A}(\hat{q}) \cap \mathcal{B}_{i=1,2,\ldots,k} = \emptyset\}$$

L'ensemble $\mathcal{CB}$ peut définir une partition de l'espace libre en plusieurs régions non connexes. Ceci a une conséquence très importante en planification de trajectoires : si l'espace libre est composé de plusieurs composantes connexes, alors il existe, à partir d'une configuration initiale donnée, un ensemble de positions du robot dans l'espace de travail qui ne sont pas accessibles [87][13].

Dans le paragraphe suivant, nous définissons les notions de chemin et de connexitépar chemin.

2.2 Chemins et ensembles chemin-connectés

Définition 2.4 (Chemin dans $\mathbb{R}^n$)
Soit I l'intervalle $[0, 1] \subset \mathbb{R}$. Un chemin *$\hat{\varphi}$ de $\mathbb{R}^n$ d'un point initial $\hat{q}_\circ$ à un point final $\hat{q}_\bullet$ est défini comme un n-uplets $(\varphi_1(\alpha), \varphi_2(\alpha), \ldots, \varphi_n(\alpha))$ de n fonctions continues de $I \to \mathbb{R}$ avec :*

$$\hat{q}_\circ = (\varphi_1(0), \varphi_2(0), \ldots, \varphi_n(0)) \quad et \quad \hat{q}_\bullet = (\varphi_1(1), \varphi_2(1), \ldots, \varphi_n(1))$$

Par conséquent, pour un robot à n degrés de liberté, toute trajectoire est représentée par un ensemble de fonctions continues dans l'espace des configurations du robot. De plus, une trajectoire sans collision $\hat{\varphi}$ ne doit pas contenir une intersection avec les $\mathcal{C}$*-obstacles*, soit : $\forall \alpha \in I$, $(\varphi_1(\alpha), \varphi_2(\alpha), \ldots, \varphi_n(\alpha)) \notin \mathcal{CB}$.

Définition 2.5 (Chemin connecté)
On dit que deux ensembles C_1 et $C_2 \subset C$ sont chemin-connectés *si et seulement si $\forall \hat{q}_1, \hat{q}_2 \in C_1, C_2$ il existe un chemin de $\hat{q}_1$ à $\hat{q}_2$ dans C.*

On peut noter qu'un ensemble chemin-connecté est forcément connexe mais que la réciproque n'est pas vraie. Par exemple, l'ensemble $[0, 1[\quad \cup \quad]1, 2]$ est connexe, mais n'est pas chemin-connecté [24].

Définition 2.6 (Espace accessible)
Soit $\hat{q}$ une configuration en $\mathcal{C}_{libre}$, on définit l'espace accessible *de $\hat{q}$: $\mathcal{C}_{libre_{\hat{q}}} \subset \mathcal{C}_{libre}$ comme l'union de tous les sous-ensembles chemin-connectés de $\mathcal{C}_{libre}$ contenant $\hat{q}$. On appelle aussi $\mathcal{C}_{libre_{\hat{q}}}$* la composante chemin connecté *de $\hat{q}$.*

Par exemple, dans la figure 2.2, nous avons deux régions non connexes et donc non chemin-connectées de l'espace libre : $\mathcal{C}_{libre_{\hat{q}_a}}$ et $\mathcal{C}_{libre_{\hat{q}_b}}$. Autrement dit, il n'existe pas de chemin en $\mathcal{C}_{libre}$ allant de $\hat{q}_a$ à $\hat{q}_b$ ou vice versa.

2.3 La distance de Hausdorff

La distance de Hausdorff [65] [24] permet de mesurer la proximité entre deux sous ensembles. Elle nous permettra de parler d'approximation de $\mathcal{C}_{libre_{\hat{q}}}$ par un ensemble discret de points.

Définition 2.7 (Distances dans $I\!R^n$)
Soit $Y \subset I\!R^n$ un espace non vide, nous définissons :

1. *la* distance entre deux points $\hat{x}, \hat{y} \in I\!R^n$: $d(\hat{x}, \hat{y}) = ||\hat{x} - \hat{y}||$,
2. *la* distance d'un point $\hat{x} \in I\!R^n$ *à l'espace* Y :
   - $d(\hat{x}, Y) = d(Y, \hat{x}) = \min\{d(\hat{x}, \hat{y}) | \hat{y} \in Y\}$ *pour* Y *énumérable,*
   - $d(\hat{x}, Y) = d(Y, \hat{x}) = \inf\{d(\hat{x}, \hat{y}) | \hat{y} \in Y\}$ *pour* Y *non énumérable,*

Définition 2.8 (Fonction et distance d'Hausdorff)
Soit $\mathcal{F}(I\!R^n)$ l'ensemble des sous-espaces non vides bornés de $I\!R^n$ et $(X, Y) \in \mathcal{F}(I\!R^n) \times \mathcal{F}(I\!R^n)$, on définit la fonction d'Hausdorff ***de*** X ***à*** Y *comme :*

$$h(X, Y) = \sup\{d(\hat{x}, Y) | \hat{x} \in X\} = \sup_{\hat{x} \in X} \inf_{\hat{y} \in Y} d(\hat{x}, \hat{y})$$

et la distance d'Hausdorff ***entre*** X ***et*** Y *comme :*

$$\rho(X, Y) = \max[h(X, Y), h(Y, X)]$$

Nous utilisons cette distance pour estimer l'approximation de tout espace X borné par un ensemble fini de points $EL \subset X$. On peut remarquer que si $EL \subset X$ alors nous avons : $\forall \hat{q}_i \in EL$, $d(\hat{q}_i, X) = 0$ (voir définition 2.7).

Ainsi, $h(EL, X) = 0$ et par conséquent :

$$\rho(EL, X) = h(X, EL)$$

La distance d'Hausdorff est une notion particulièrement importante en planification de trajectoires. En effet, si pour tout ε nous sommes capables de construire, à partir d'une configuration donnée $\hat{q}_\circ \in \mathcal{C}_{libre}$, un ensemble EL de points accessibles depuis $\hat{q}_\circ$ tel que :

$$\rho(\mathcal{C}_{libre_{\hat{q}_\circ}}, EL) \leq \varepsilon$$

alors pour toute configuration $\hat{q} \in \mathcal{C}_{libre_{\hat{q}_\circ}}$ nous avons $d(\hat{q}, EL) \leq \varepsilon$. Autrement dit, nous pouvons placer un point arbitrairement près de tout point de l'espace accessible.

2.4 Billage et pavage d'un espace

Dans le paragraphe précédent nous avons expliqué comment il était possible d'estimer l'approximation de l'espace accessible d'une configuration $\hat{q}_\circ \in \mathcal{C}_{libre}$ par un ensemble fini EL de points. Si nous sommes capables de construire une telle approximation, alors nous avons la caractéristique suivante : pour toute configuration $\hat{q} \in \mathcal{C}_{libre_{\hat{q}_\circ}}$ il existe au moins un point $\hat{p} \in EL$ tel que la distance entre $\hat{q}$ et $\hat{p}$ est inférieure à ε. Autrement dit, toute configuration $\hat{q} \in \mathcal{C}_{libre_{\hat{q}_\circ}}$ possède au moins un représentant en EL à $\varepsilon-$près.

De cette façon au lieu de calculer tout l'espace libre du robot on peut l'approcher par un ensemble fini de points. Chacun de ces points représente les configurations voisines à une distance inférieure ou égale à ε. Ceci revient à réaliser un *pavage* à $\varepsilon-$près de l'espace accessible de $\hat{q}_\circ$.

Bien entendu, nous sommes intéressés par des pavages "efficaces", c'est-à-dire que nous cherchons à approximer $\mathcal{C}_{libre_{\hat{q}_\circ}}$ avec le plus petit nombre de points. Ainsi, il faut établir un éloignement minimal entre les représentants. Cet éloignement définit implicitement un *billage* de l'espace approché.

Avant de définir formellement les concepts de billage et de pavage [72], nous ferons un rappel de quelques concepts fondamentaux. A partir de ces concepts nous définirons la borne supérieure de la cardinalité du billage de rayon r d'un espace, autrement dit la borne supérieure du nombre de boules de rayon r pouvant occuper un sous espace borné de $\mathbb{R}^n$.

Définition 2.9 (Ensemble convexe)
On appelle un ensemble convexe *tout domaine D dans l'espace Euclidien de dimension n (noté $\mathbb{E}^n$) tel que pour tout $\hat{p}_i, \hat{p}_j \in D$ le segment $\overline{\hat{p}_i\hat{p}_j}$ soit complètement contenu en D, c'est-à-dire, $\forall \alpha \in [0,1]$, $\hat{p} = (\hat{p}_i * (1 - \alpha) + \hat{p}_j * \alpha) \in D$.*

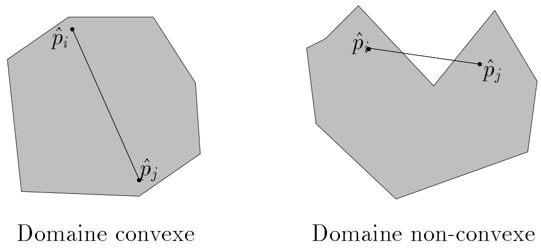


FIG. 2.3 - Convexité d'un espace.

Définition 2.10 (Enveloppe convexe)
*L'*enveloppe convexe *d'un ensemble de points S de $\mathbb{E}^n$ est le plus petit domaine convexe de $\mathbb{E}^n$ contenant S.*

Définition 2.11 (Simplex)
Un simplex *S est défini comme l'enveloppe convexe d'un ensemble de $(n+1)$ points indépendants en $\mathbb{E}^n$. On dit que S est* régulier *si tous ses côtés sont égaux .*

Définition 2.12 (Boule en $\mathbb{E}^n$)
L'ensemble $Bl(\hat{x}, r) \subset \mathbb{E}^n$ est appelé "boule de centre $\hat{x}$ et de rayon r", *et il est défini comme $Bl(\hat{x}, r) = \{\hat{y} \in \mathbb{R}^n \ \mid \ ||\hat{x} - \hat{y}|| \leq r\}$.*

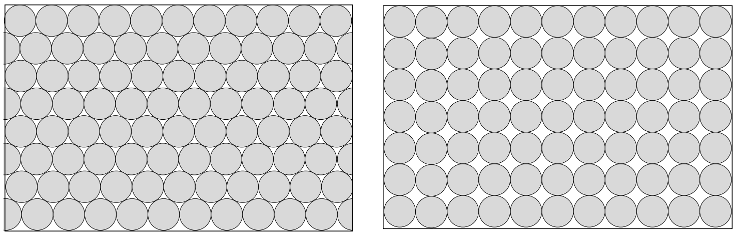
Ainsi, soit EL un ensemble de points de $\mathbb{E}^n$. L'*ensemble de boules de rayon r produit par EL* est noté $U(EL, r)$ c'est-à-dire :

$$U(EL, r) = \{Bl(\hat{p}, r) \ \mid \ \hat{p} \in EL\}$$

Définition 2.13 (Billage)
Soit $EL = (\hat{p}_i)_{i \in \mathbb{N}^*}$ une séquence de points de $\mathbb{E}^n$ et $r \in \mathbb{R}$. $\mathcal{K} = U(EL, r)$ est appelé un billage *si pour tout couple de points $(\hat{p}_i, \hat{p}_j) \in EL \times EL$ où $i \neq j$ $d(\hat{p}_i, \hat{p}_j) \geq 2r$. Nous appelons EL ensemble des balises et r rayon du billage.*

Nous notons par C_s **le cube** $\{\hat{x} \in \mathbb{R}^n | -\frac{s}{2} \leq x_i \leq \frac{s}{2}, 1 \leq i \leq n\}$ et par $\nu(Y)$ **le volume** d'un sous-ensemble compact[1] Y de $\mathbb{R}^n$.

[1]fermé et borné.



a b

FIG. 2.4 - Deux billages de densité différente en $\mathbb{R}^2$.

Ainsi, nous définissons :

Définition 2.14 (Densité d'un billage)
Soit $\mathcal{K}$ *un billage en* $\mathbb{R}^n$*, nous définissons la* densité du billage $\mathcal{K}$ *comme :*

$$\Delta_n(\mathcal{K}) = \lim_{s\to\infty} \frac{\sum_{\hat{p}_i : Bl(\hat{p}_i, r)\subset C_s} \nu(Bl(\hat{p}_i, r))}{\nu(C_s)} \tag{2.1}$$

La figure 2.4 montre deux billages dans $\mathbb{E}^2$ avec des densités différentes.

Il est facile de voir que la borne supérieure de la densité d'un billage est indépendante du rayon des boules utilisées. La meilleure estimation de la *borne supérieure* pour un billage dans $\mathbb{E}^n$ notée σ_n a été établie par Rogers [69]. La borne supérieure est définie comme la proportion du volume d'un simplex régulier S de côté 2 dans $\mathbb{E}^n$, lequel est couvert par les $(n+1)$ boules de rayon 1, centrées aux sommets de S (voir figure 2.5). σ_n est appelée la densité de Rogers. La figure 2.5 représente les simplex réguliers pour $\mathbb{E}^2$ et $\mathbb{E}^3$. Par exemple, dans le cas du simplex régulier en $\mathbb{E}^2$ il est très facile de calculer la densité de Rogers. L'aire du triangle est $\sqrt{3}$. L'aire de chacun des secteurs (marqués en gris dans la figure 2.5) est $\pi/6$. Ainsi, l'aire des trois secteurs est $3\pi/6 = \pi/2$. La densité de Rogers pour $\mathbb{E}^2$ est alors $\pi/\sqrt{12} \approx 0.9069$ [72].

A partir de la borne supérieure de la densité de Rogers nous pouvons introduire la borne supérieure de la *densité centrale* de Rogers [73] notée σ_n' et définie comme :

$$\sigma_n' = \sigma_n / J_n \tag{2.2}$$

où J_n est le volume d'une boule de rayon 1 dans $\mathbb{E}^n$. Le volume J_n est égal à :

$$J_n = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)}$$

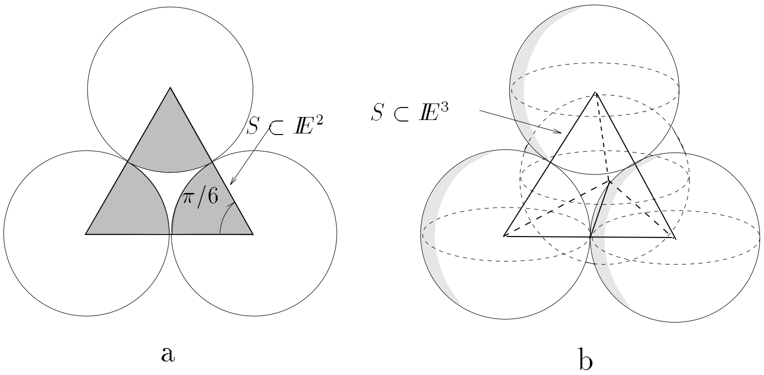


FIG. 2.5 - Un simplex régulier en $\mathbb{E}^2$ et un autre en $\mathbb{E}^3$.

où la fonction Γ est définie comme suit :

$$\Gamma(a) = \int_0^\infty e^{-x} x^{a-1} dx$$

Nous pouvons calculer $\Gamma(a)$ avec les règles suivantes :

$$\Gamma(0) = 1, \Gamma(1) = 1$$
$$\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}, \quad \Gamma(n) = (n-1)!$$
$$\Gamma(a) = (a-1)\Gamma(a-1)$$

Ainsi, pour une boule de rayon r en $\mathbb{E}^n$ nous avons :

$$\nu(Bl(\hat{x}, r)) = r^n J_n$$

Dans le cas de l'exemple précédent, on peut donc calculer la densité centrale de Rogers (pour $n = 2$), $J_2 = \pi$. D'où $\sigma' = (\pi/\sqrt{12})/\pi \approx .2886$ [72].

Pour un ensemble $X \subset \mathbb{R}^n$ nous notons $s(X) = \min\{s \in \mathbb{R} | X \subset C_s\}$, c'est-à-dire que $s(X)$ est la taille du côté du plus petit cube contenant X.

On remarque ainsi que si nous avons un billage $\mathcal{K} = U(EL, r)$ d'un espace $X \subset \mathbb{R}^n$ avec la propriété $EL \subset X$ alors $\mathcal{K} \subset C_{(s(X)+2r)}$ (voir figure 2.6). Nous pouvons alors montrer la proposition suivante :

Proposition 2.1 (Borne supérieure de la cardinalité d'un billage) *Soit $X \subset \mathbb{R}^n$ et $\mathcal{K} = U(EL, r)$ un billage sphérique tel que $EL \subset X$ alors la borne supérieure de la cardinalité de EL est donnée par :*

$$Card(EL) \leq \left\lfloor \frac{{\sigma_n}' * (s(X) + 2r)^n}{r^n} \right\rfloor \quad (2.3)$$

Démonstration :

Soit $\mathcal{K}' = U(EL', r)$ le billage de cardinalité maximale de $C_{s(X)}$ tel que $EL' \subset C_{s(X)}$. La densité Δ de ce billage est :

$$\Delta = \frac{Card(EL') * J_n * r^n}{(s(X) + 2r)^n} \equiv \frac{volume\ des\ boules}{volume\ du\ cube}$$

ainsi,

$$Card(EL') = \frac{\Delta * (s(X) + 2r)^n}{J_n * r^n}$$

étant donné que $\Delta \leq \sigma_n$ et en utilisant l'expression 2.2 nous avons :

$$Card(EL') \leq \left\lfloor \frac{\sigma_n * (s(X) + 2r)^n}{J_n * r^n} \right\rfloor = \left\lfloor \frac{{\sigma_n}' * (s(X) + 2r)^n}{r^n} \right\rfloor$$

Etant donné que $X \subset C_{s(X)}$ nous avons : $Card(EL) \leq Card(EL')$. □

Définition 2.15 (Pavage) *Soit $EL = (\hat{p}_i)_{i \in \mathbb{N}^*}$ une séquence de points de $\mathbb{R}^n$ et $r \in \mathbb{R}$ alors $P = U(EL, r)$ est appelé un* pavage *de X si et seulement si $X \subset P$. Nous appellons EL l'ensemble de balises et r le rayon du pavage.*

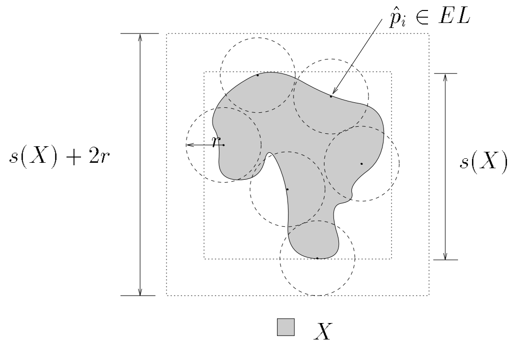


FIG. 2.6 - Billage d'un espace X et le cube $C_{s(X)}$.

Une conséquence immédiate de la définition d'un pavage est que $\rho(EL, X) \leq r$ et par conséquent EL est une approximation de X à r-près au sens de la distance de Hausdorff.

Dans le chapitre qui suit, nous présentons une méthode de construction d'un ensemble de points $EL \subset \mathcal{C}_{libre_{\hat{q}_0}}$ qui réalise à la fois un billage $\mathcal{K} = U(EL, \varepsilon/2)$ et un pavage $P = U(EL, \varepsilon)$ de l'espace libre. Le pavage P de $\mathcal{C}_{libre_{\hat{q}_0}}$ nous permet de montrer que EL est une approximation de $\mathcal{C}_{libre_{\hat{q}_0}}$ et le billage K nous permet d'affirmer que cette approximation peut être obtenue en un nombre fini d'étapes.

Chapitre 3

L'algorithme $EXPLORE$

3.1 L'algorithme $EXPLORE_\infty$

L'algorithme $EXPLORE_\infty$ est utilisé afin de construire incrémentalement un pavage $\mathcal{K}$ pour tout espace compact $X \subset I\!R^n$. Le principe de la méthode consiste à placer des *balises* dans l'espace X en commençant par un point initial donné $\hat{x}_\circ \in X$. Ainsi, la première balise L_1 de l'ensemble de balises EL est le point $\hat{x}_\circ$. Puis, $EXPLORE_\infty$ sélectionne le point $\hat{p} \in X$ le plus éloigné de $\hat{x}_\circ$, qui devient alors une nouvelle balise de EL. On continue cette procédure en sélectionnant à chaque itération le point de X le plus éloigné possible des balises précédemment posées. De cette façon, si chacune des balises est posée à une distance supérieure ou égale à r des balises précédemment posées, on est certain de réaliser un billage de l'espace libre avec des boules de rayon $r/2$. En utilisant la densité de Rogers on montre que le nombre de balises espacées de $r/2$ est borné et donc que l'on va pouvoir approximer l'espace libre à r près en un temps fini. Si le nombre de balises continue à augmenter, le rayon r ne peut que diminuer ; ce qui conduit nécessairement à une approximation de plus en plus fine de l'espace libre.

L'algorithme $EXPLORE_\infty$ est présenté ci dessous. Il a pour paramètres d'entrée la valeur de résolution souhaitée $\varepsilon \in I\!R$, et $\hat{x}_\circ \in X$ le point de départ. L'algorithme $EXPLORE_\infty$ nous retourne EL^ε, qui est à la fois l'ensemble des balises du billage à $\varepsilon'/2$-près de X et l'ensemble de balises du pavage de rayon ε', où $\varepsilon' \leq \varepsilon$.

L'algorithme est le suivant :

$ALGORITHME\_EXPLORE_\infty(\hat{x}_\circ, \varepsilon)$
begin
  $L_1 := \hat{x}_\circ$;
  $EL_1 := \{L_1\}$
  $t := 1$;
  do
    $\hat{p}_t := \hat{p} : \max_{\hat{p} \in X} d(EL_t, \hat{p})$;
    $L_{t+1} := \hat{p}_t$;   /* on choisit la nouvelle balise */
    $EL_{t+1} := EL_t \cup \{L_{t+1}\}$;
    $\varepsilon_t := d(EL_t, \hat{p}_t)$;
    $t := t + 1$
  while($\varepsilon_{t-1} > \varepsilon$); /* ε' est ε_{t-1} */
  $EL^\varepsilon := EL_{t-1}$;
  return(EL^ε);
end

L'algorithme commence de la manière suivante : soit $\hat{x}_\circ \in X$ le point initial donné, nous initialisons $EL_1 = \{L_1\}$ où $L_1 = \hat{x}_\circ$, et nous appelons $\hat{p}_1$ le point tel que :

$$\hat{p}_1 : \max_{\hat{p} \in X} d(L_1, \hat{p})$$

En accord avec notre définition, $L_2 = \hat{p}_1$ est le point le plus éloigné dans X de L_1. On pose $EL_2 = EL_1 \cup \{L_2\}$ et $\varepsilon_1 = \max_{\hat{p} \in X} d(L_1, \hat{p})$ pour finir la première itération. Il est clair que $U(EL_2, \varepsilon_1/2)$ est un billage de X et que $U(EL_2, \varepsilon_1)$ est un pavage de X, c'est-à-dire $X \subset U(EL_2, \varepsilon_1)$. Nous continuons avec la deuxième itération de l'algorithme : étant donné un point $\hat{p} \in X$ nous considérons le minimum entre $d(L_1, \hat{p})$ et $d(L_2, \hat{p})$ c'est-à-dire $d(EL_2, \hat{p})$ et nous essayons de maximiser cette valeur entre tous les points de X :

$$\hat{p}_2 : \max_{\hat{p} \in X} d(EL_2, \hat{p}) = \max_{\hat{p} \in X} \min_{L_i \in EL_2} d(L_i, \hat{p})$$

La troisième balise est alors $L_3 = \hat{p}_2$; $EL_3 = \{L_1, L_2, L_3\}$ et $\varepsilon_2 = \max_{\hat{p} \in X} d(EL_2, \hat{p})$. De même qu'à la première itération, $U(EL_3, \varepsilon_2/2)$ et $U(EL_3, \varepsilon_2)$ représente respectivement un billage et un pavage de X. D'une manière générale, pour un ensemble

EL_t avec t balises nous cherchons à optimiser $\max_{\hat{p}\in X} d(EL_t, \hat{p})$. Dès lors nous notons pour $t \in I\!N^+$:

$$\hat{p}_t : \max_{\hat{p}\in X} d(EL_t, \hat{p}) = \max_{\hat{p}\in X} \min_{L_i \in EL_t} d(L_i, \hat{p})$$

En posant $L_{t+1} = \hat{p}_t$ nous obtenons notre $(t+1)^{ième}$ balise. Les ensembles $U(EL_{t+1}, \varepsilon_t/2)$ et $U(EL_{t+1}, \varepsilon_t)$ sont respectivement un billage et un pavage de l'ensemble X.

Ainsi, chaque itération t se ramène à un problème d'optimisation. Etant donné une itération $t \in I\!N^+$, $EXPLORE_\infty$ est une fonction de la forme[1] :

$$EXPLORE_\infty(t) = \begin{cases} \max d(EL_t, \hat{p}) \\ \hat{p} \in X \end{cases}$$

Etant donné que X est un espace borné, nous avons le théorème suivant :

Théorème 3.1 (Convergence)
Soit X un espace compact, nous avons pour l'application de $EXPLORE_\infty$ en X :

$$\lim_{t\to\infty} EXPLORE_\infty(t) = 0 \qquad (3.4)$$

Démonstration :

Soit $t \in I\!N^+$ et $EXPLORE_\infty(t) = \varepsilon_t$, il est évident que $\mathcal{K}_{t+1} = U(EL_{t+1}, \varepsilon_t/2)$ est un billage de X. De cette manière, si $s = s(X)$ nous obtenons :

$$\sigma_n \geq \Delta_n(\mathcal{K}_{t+1}) = \frac{(t+1) * J_n * (\varepsilon_t/2)^n}{(s+\varepsilon_t)^n} \equiv \frac{\textit{volume des boules}}{\textit{volume du cube}}$$

ainsi,

$$\left(\frac{\varepsilon_t}{2s + 2\varepsilon_t}\right)^n \leq \frac{\sigma_n{}'}{t+1}$$

de là nous avons :

[1] Attention, il faut distinguer la procédure $ALGORITHME\_EXPLORE_\infty(\hat{x}_\circ, \varepsilon)$ de la fonction $EXPLORE_\infty(t)$

$$\varepsilon_t \leq 2 * \left(\frac{\sigma'}{t+1}\right)^{\frac{1}{n}} * (s + \varepsilon_t)$$

donc,

$$\varepsilon_t * \left(1 - 2 * \left(\frac{\sigma'}{t+1}\right)^{\frac{1}{n}}\right) \leq 2s * \left(\frac{\sigma'}{t+1}\right)^{\frac{1}{n}}$$

ainsi pour $(1 - 2 * (\sigma'/(t+1))^{\frac{1}{n}}) > 0$*, c'est-à-dire pour* $t > \lceil 2^n {\sigma_n}' - 1 \rceil$*, nous avons le résultat suivant :*

$$\varepsilon_t \leq \frac{2s * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}}{1 - 2 * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}} \qquad (3.5)$$

Par conséquent :

$$\lim_{t\to\infty} EXPLORE_\infty(t) \leq \lim_{t\to\infty} \frac{2s * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}}{1 - 2 * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}}$$

$$\lim_{t\to\infty} \frac{2s * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}}{1 - 2 * (\frac{{\sigma_n}'}{t+1})^{\frac{1}{n}}} = 0$$

Ce qui nous donne le résultat cherché. □

En utilisant la proposition 2.1, on peut explicitement donner le nombre d'itérations T qui rend $EXPLORE_\infty(T) < \varepsilon$. Ce nombre T d'itérations est relié à la borne supérieure de la cardinalité d'un billage de rayon $\varepsilon/2$ de l'espace X.

Théorème 3.2 (Nombre fini d'itérations)

$$Si \quad T > \left\lfloor \frac{{\sigma_n}' * (s(X) + \varepsilon)^n}{(\frac{\varepsilon}{2})^n} \right\rfloor - 1 \quad alors \quad EXPLORE_\infty(T) < \varepsilon. \qquad (3.6)$$

De ce fait, nous pouvons réaliser un pavage de X dont le rayon tend vers 0 et donc approcher tout point de X à une distance ε arbitraire. Dans la suite de l'exposé, nous proposons un algorithme similaire pour la construction d'un pavage d'un sous-ensemble particulier de l'espace $\mathcal{C}_{libre}$: *l'espace libre à* ε*−près* noté $\mathcal{C}^{\varepsilon}_{libre}$. Cette construction est effectuée à partir d'une configuration initiale $\hat{q}_\circ$ donnée en utilisant des trajectoires d'un type particulier : les chemins Manhattan.

3.2 Espace libre à ε-près et chemins Manhattan

Dans cette partie nous définissons la notion d'espace libre à $\varepsilon-$près. Intuitivement, on peut dire qu'une trajectoire dans cet espace passe toujours à plus de ε des $\mathcal{C}$-*obstacles*. Nous introduisons ensuite un type particulier de trajectoire : les chemins Manhattan d'ordre k. Sur un tel chemin on ne déplace qu'un degré de liberté à la fois. L'intérêt de ces chemins est double : ils sont facilement paramètrables par $n*k$ réels (n est la dimension de l'espace des configuration considéré) et ils facilitent les calculs géométriques. Puis nous présentons l'algorithme $EXPLORE$ ainsi que les résultats principaux qui forment le cœur de ce chapitre, à savoir :

1. Si un point est dans la même composante chemin-connecté de l'espace libre à ε-près que la position initiale, alors on peut, en un nombre fini d'itérations de l'algorithme $EXPLORE$, trouver un chemin Manhattan partant de l'origine qui s'approche à $\varepsilon-$près de ce point. Autrement dit on peut se déplacer partout (à ε-près) dans l'espace accessible avec des chemins Manhattan de longueur finie.

2. La procédure permettant de réaliser cette exploration est basée sur la maximisation d'un fonctionnelle de $\mathbb{R}^{n*k}$ à valeur dans $\mathbb{R}^+$ et peut donc être réalisée en pratique sur un ordinateur.

3.2.1 L'espace libre à $\varepsilon-$près

Définition 3.1 (Espace libre à $\varepsilon-$près)
Etant donné $\varepsilon > 0$ et la métrique $d(\hat{x}, \hat{y})$, l'espace libre à $\varepsilon-$près *de $\mathcal{C}_{libre}$ est défini par :*

$$\mathcal{C}^{\varepsilon}_{libre} = \{\hat{q} \in \mathcal{C}_{libre} | d(\hat{q}, \mathcal{CB}) \geq \varepsilon\}.$$

Un exemple de cet espace en $\mathbb{R}^2$ est montré dans la figure 3.1. Evidemment $\mathcal{C}^{\varepsilon}_{libre}$ peut être composé d'une ou plusieurs composantes connexes bornées.

3.2.2 Les Chemins Manhattan

Soit $n \in \mathbb{N}^+$. On note $\wp(n)$ l'ensemble des fonctions continues de $I = [0, 1]$ à valeurs dans $\mathbb{R}^n$. Autrement dit :

$$\hat{\varphi} \in \wp(n) \Longleftrightarrow \begin{cases} \hat{\varphi} : [0,1] \to \mathbb{R}^n \\ \alpha \to [\varphi_1(\alpha), \ldots, \varphi_n(\alpha)] \end{cases}$$

où chaque fonction φ_i, $i = 1, 2, \ldots n$ est continue de $[0,1]$ dans $I\!R$. Comme nous allons le voir, $\wp(n)$ est un espace vectoriel normé.

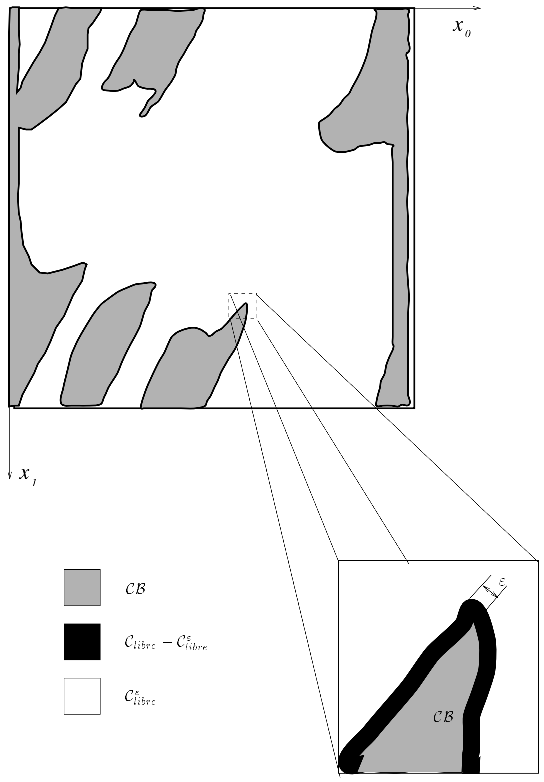


FIG. 3.1 - L'espace $\mathcal{C}^{\varepsilon}_{libre}$.

Un *chemin* de $I\!R^n$ est, d'un "point de vue géométrique", l'ensemble image $\hat{\varphi}([0,1]) = \{\hat{\varphi}(\alpha), \alpha \in [0,1]\}$ pour $\hat{\varphi} \in \wp(n)$. Il est clair qu'un tel chemin $\hat{\varphi}$ en $I\!R^n$ admet plusieurs paramètrisations, par exemple $sin(\alpha * 2\pi) = cos(\alpha * 2\pi + \frac{\pi}{2})$ pour $\alpha \in I$. De façon plus précise, on définit :

Définition 3.2 (Relation d'équivalence, classe d'équivalence)

1. *Soit $\hat{\varphi}$ et $\hat{\tau} \in \wp(n)$. Alors $\hat{\varphi} \sim \hat{\tau}$ si et seulement si :*
   (a) $\hat{\varphi}(0) = \hat{\tau}(0)$ *et* $\hat{\varphi}(1) = \hat{\tau}(1)$
   (b) $\exists f : [0,1] \rightarrow [0,1]$ *tel que* f *est continue et* $\forall \alpha \in [0,1]$, $\hat{\varphi}(\alpha) = \hat{\tau}(f(\alpha))$.

   $\hat{\varphi} \sim \hat{\tau}$ *définit une* relation d'équivalence *dans* $\wp(n)$.
2. *Un* chemin *de* $I\!R^n$ *est une* classe d'équivalence *dans* $\wp(n)$ *modulo la relation d'équivalence* $\hat{\varphi} \sim \hat{\tau}$, *c'est-à-dire un élément de* $\wp(n)/\sim$.

En pratique, nous identifions les chemins de $\mathbb{R}^n$ avec les fonctions de $\wp(n)$, c'est-à-dire avec leurs représentants. De cette façon deux fonctions avec la même image sont considérées comme le même chemin. On peut remarquer que cette dernière définition est plus générale que celle donnée dans la définition 2.4.

Introduisons une norme sur l'espace vectoriel $\wp(n)$. Pour $\hat{\varphi} \in \wp(n)$, on pose :

$$||\hat{\varphi}||_\infty = \max\{||\varphi_i||_\infty\} \quad \text{pour} \quad i \in \{1, 2, \ldots, n\}$$

où

$$||\varphi_i||_\infty = \sup\{|\varphi_i(\alpha)|_\infty\} \quad t \in [0, 1]$$

Ainsi, $(\wp(n), ||\ ||_\infty)$ devient un espace vectoriel normé.

Définition 3.3 (Fonction Manhattan)
On considère le sous-ensemble $\mathcal{M}(n)$ de $\wp(n)$ des fonctions $\hat{\gamma} : [0,1] \to \mathbb{R}^n$, $\alpha \to (\gamma_1(\alpha), \ldots \gamma_n(\alpha))$ défini de la manière suivante :
$\hat{\gamma} \in \mathcal{M}(n)$ si et seulement s'il existe une subdivision

$$(c_j)_{j=0,1,\ldots,N} \quad de \quad [0,1] : 0 = c_0 < c_1 < \ldots < c_N = 1$$

telle que : $\forall j \in \{0, 1, \ldots, N-1\}, \exists k_j \in \{1, 2, \ldots n\}$ où la fonction numérique γ_{k_j} soit affine sur $[c_j, c_{j+1}]$ et chaque γ_i, $i \neq k_j$ soit constante sur $[c_j, c_{j+1}]$

Autrement dit sur $[c_j, c_{j+1}]$ seule la fonction γ_{k_j} peut être non constante (et affine). Une telle fonction $\hat{\gamma}$ est appelée fonction Manhattan.

Un exemple d'une fonction $\hat{\varphi} \in \wp(2)$ et une fonction Manhattan $\hat{\gamma} \in \mathcal{M}(2)$ sont montrées dans la figure 3.2. Les images des deux fonctions, $\hat{\varphi}([0,1])$ et $\hat{\gamma}([0,1])$ sont montrées dans la figure 3.3. Ainsi, la fonction $\hat{\varphi}$ est composée de deux fonctions continues $\gamma_1 : I \to \mathbb{R}$ et $\gamma_2 : I \to \mathbb{R}$. Comme nous l'avons défini précédemment seule une des deux fonctions peut être non constante dans chacun des intervalles formés par les subdivisions $(c_j)_{j=0,1,\ldots N}$. La figure 3.4 montre cette situation.

Comme précédemment, on définit géométriquement un *chemin Manhattan* comme l'ensemble image $\hat{\gamma}([0,1])$ ou comme une classe d'équivalence[2] modulo la relation $\sim$.

[2] On peut montrer que $\mathcal{M}(n)$ est dense dans $\wp(n)$. Autrement dit, on peut approcher toute courbe paramétrée continue, c'est-à-dire tout chemin de $\mathbb{R}^n$, d'aussi près que l'on veut par un chemin Manhattan. Soit :

$$\forall \hat{\varphi} \in \wp(n), \forall \varepsilon > 0, \exists \hat{\gamma} \in \mathcal{M}(n) : ||\hat{\varphi} - \hat{\gamma}||_\infty < \varepsilon$$

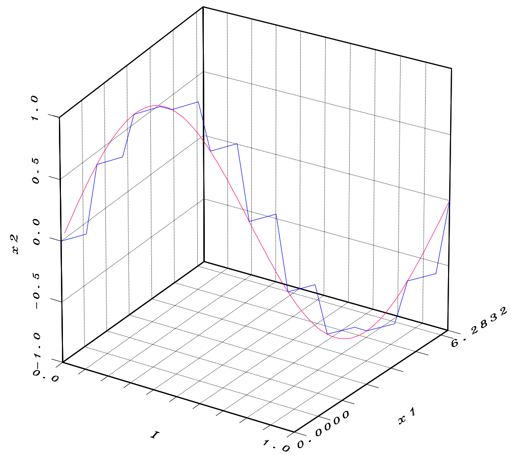


FIG. 3.2 - Une fonction $\hat{\varphi} \in \wp(2)$ et une autre $\hat{\gamma} \in \mathcal{M}(2)$

On peut alors conclure que se déplacer sur un chemin Manhattan dans l'espace des configurations d'un robot à n degrés de liberté revient à se déplacer selon une dimension à la fois, ce qui est équivalent à bouger successivement chacun des degrés de liberté.

Définition 3.4 (Chemin Manhattan d'ordre 1)
Soit un système à n degrés de liberté et $\hat{q}_\circ = (x_1, \ldots, x_n)$ une configuration donnée de ce système, on définit un chemin Manhattan d'ordre 1 *et partant de $\hat{q}_\circ$ comme l'application $\hat{\gamma}^1 : \alpha \rightarrow I\!R^n$ où :*

$$\gamma_i(\alpha) = \begin{cases} x_i & pour \quad 0 \le \alpha \le \frac{(i-1)}{n} \\ x_i + \Delta_i * (n * \alpha - i + 1) & pour \quad \frac{(i-1)}{n} \le \alpha \le \frac{i}{n} \\ x_i + \Delta_i & pour \quad \frac{i}{n} \le \alpha \le 1 \end{cases} \qquad (3.7)$$

avec $\Delta_i \in I\!R$. Ce type de trajectoire consiste à déplacer successivement un degré de liberté à la fois, chacun à son tour. De cette manière nous définissons $\hat{\gamma}^1$ comme :

$$\hat{\gamma}^1 = (\Delta_1^1, \Delta_2^1, \ldots, \Delta_i^1, \ldots, \Delta_n^1) \qquad (3.8)$$

Définition 3.5 (Concaténation de deux chemins)
La concaténation *de deux chemins $\hat{\varphi}$ et $\hat{\tau}$ appartenant à $\wp(n)$ est définie seulement dans le cas où $\hat{\varphi}(1) = \hat{\tau}(0)$. Elle est définie comme suit :*

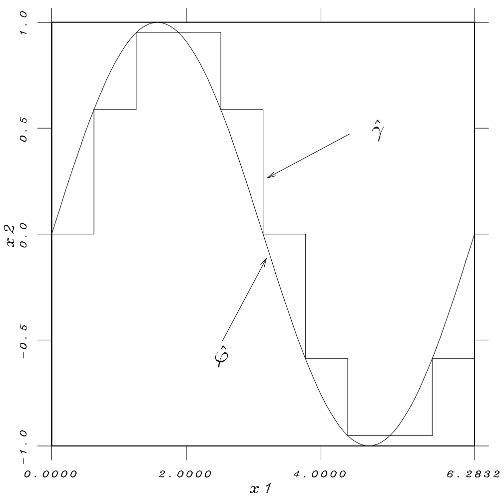


FIG. 3.3 - Image de $\hat{\varphi}$ et $\hat{\gamma}$.

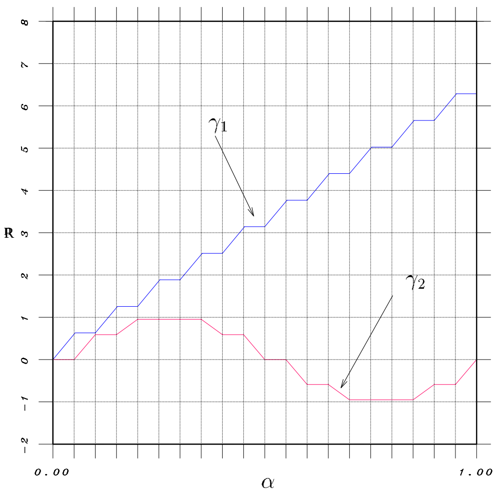


FIG. 3.4 - Les fonctions γ_1 et γ_2.

$$\hat{\varphi} * \hat{\tau} = \begin{cases} \hat{\varphi}(2t) & \textit{pour } 0 \leq t \leq 1/2 \\ \hat{\tau}(2t-1) & \textit{pour } 1/2 \leq t \leq 1 \end{cases}$$

Définition 3.6 (Chemin Manhattan d'ordre k)
Soit un système à n degrés de liberté et $\hat{q}_\circ = (x_1, \ldots, x_n)$ une configuration donnée de ce système. On définit un chemin Manhattan d'ordre k *partant de $\hat{q}_\circ$ comme la concaténation de k chemins Manhattan d'ordre 1. On note un tel chemin :*

$$\hat{\gamma}^k = \hat{\gamma}_1^1 * \hat{\gamma}_2^1 * \ldots * \hat{\gamma}_k^1 = (\Delta_1^1, \Delta_2^1, \ldots, \Delta_n^1, \ldots, \Delta_i^j, \ldots, \Delta_n^k) \qquad (3.9)$$

Où Δ_i^j représente le mouvement relatif du degré de liberté i dans $\hat{\gamma}_j^1$. Ainsi, $\hat{\gamma}_0^1(0) = \hat{q}_\circ$,$\hat{\gamma}_k^1(1) = \hat{q}_\bullet$ et $\hat{\gamma}_{j-1}^1(0) = \hat{\gamma}_j^1(1)$ pour $j = 2, \ldots, k$. Par exemple, la figure 3.5 représente un chemin Manhattan d'ordre 5 pour un robot à deux degrés de liberté.

Au vu de cette dernière définition, tout chemin Manhattan $\hat{\gamma}^k$ d'ordre k d'une configuration initiale $\hat{q}_\circ$ à une configuration finale $\hat{q}_\bullet$ est décrit par un ensemble

$M(\hat{\gamma}^k) = \{\hat{\gamma}_1^1, \hat{\gamma}_2^1, \ldots, \hat{\gamma}_k^1\}$ de chemins Manhattan de dimension 1, plus précisément :

$$\hat{\gamma}^k(t) = \hat{\gamma}_1^1 * \hat{\gamma}_2^1 * \ldots * \hat{\gamma}_k^1 = \begin{cases} \hat{\gamma}_1^1(t * k) & \text{pour} \;\; 0 \leq t \leq \frac{1}{k} \\ \hat{\gamma}_2^1((t - \frac{1}{k}) * k) & \text{pour} \;\; \frac{1}{k} \leq t \leq \frac{2}{k} \\ \vdots & \\ \hat{\gamma}_i^1((t - \frac{i-1}{k}) * k) & \text{pour} \;\; \frac{i-1}{k} \leq t \leq \frac{i}{k} \\ \vdots & \\ \hat{\gamma}_k^1((t - \frac{k-1}{k}) * k) & \text{pour} \;\; \frac{k-1}{k} \leq t \leq 1 \end{cases}$$

3.3 L'algorithme $EXPLORE$

Avant de définir l'algorithme $EXPLORE$ en utilisant des chemins Manhattan, nous définissons la post-image produite par ce type de chemins.

Définition 3.7 (Ensemble de chemins Manhattan)
Soit $\mathcal{C}_{libre}$ l'espace des configurations d'un système à n degrés de liberté et $\hat{q} \in \mathcal{C}_{libre}$ une configuration de cet espace. On note $\mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q})$ l'ensemble de chemins Manhattan *d'ordre $k \leq \ell$ où pour tout $\hat{\gamma}^k \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}), \hat{\gamma}(0) = \hat{q}$ et $\hat{\gamma}^k([0,1]) \subset \mathcal{C}_{libre}$.*

Définition 3.8 (Post-image produite par les chemins Manhattan)
Etant donné un ensemble de balises EL de l'espace libre d'un système à n degrés de liberté nous définissons la "post-image" *de EL produite par $\mathcal{M}(\mathcal{C}_{libre}; \ell, EL)$ comme :*

$$\mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre}; \ell, EL) = \{\hat{q} = \hat{\gamma}(1) | \hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, EL)\}$$

c'est-à-dire l'ensemble de toutes les configurations en $\mathcal{C}_{libre}$ pour lequelles il existe un chemin Manhattan d'ordre inférieur ou égal à ℓ et ayant comme point de départ une balise $L_i \in EL$.

Dans la suite nous omettrons l'indice "k" pour tout chemin Manhattan d'ordre $k \leq \ell$ en $\mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q})$. C'est-à-dire que nous l'écrirons uniquement $\hat{\gamma}$.

3.3.1 Description de l'algorithme

Nous définissons l'algorithme $EXPLORE$ comme une variante de l'algorithme $EXPLORE_\infty$ La différence réside dans l'espace de recherche utilisé par la fonction

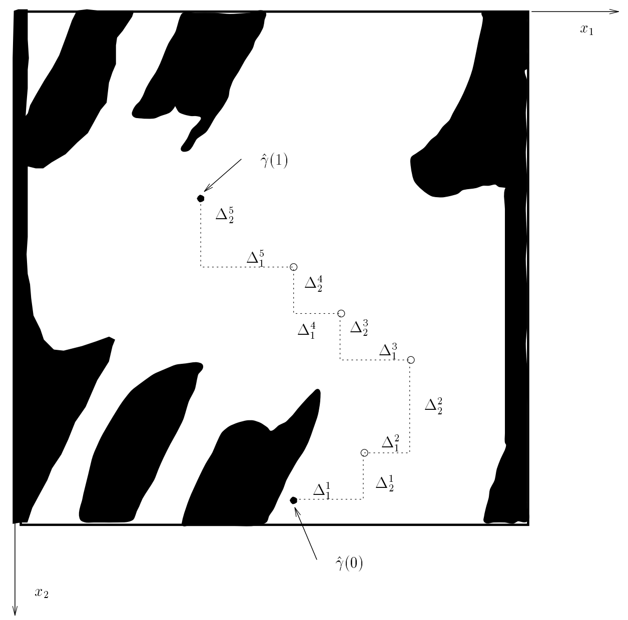


FIG. 3.5 - Un chemin Manhattan dans l'espace des configurations.

d'optimisation de $EXPLORE$. Cet espace est la post-image engendrée par l'ensemble des trajectoires Manhattan d'ordre inférieur ou égal à une valeur ℓ donnée. Ainsi, l'algorithme est le suivant :

$ALGORITHME\_EXPLORE(\hat{q}_\circ, \varepsilon)$
begin
  $L_1 := \hat{q}_\circ;$
  $EL_1 := \{L_1\}$
  $t := 1;$
  do
    $\hat{q}_t := \hat{q} : \max_{\hat{q} \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre};\ell,EL_t)} d(EL_t, \hat{q});$
    $L_{i+1} := \hat{p}_i;$
    $EL_{t+1} := EL_t \cup \{L_{t+1}\};$
    $\varepsilon_t := d(EL_t, \hat{q}_t);$
    $t := t + 1$
  while$(\varepsilon_{t-1} > \varepsilon);$
  $EL^\varepsilon := EL_{t-1}\,;$
  return$(EL^\varepsilon);$
end

Si nous avons t balises nous notons :

$$\hat{q}_t : \max_{\hat{q} \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre};\ell,EL_t)} d(\hat{q}, EL_t) = \max_{\hat{q} \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre};\ell,EL_t)} \min_{L_i \in EL_t} d(L_i, \hat{q})$$

la configuration $\hat{q}_t$ définit la nouvelle balise $L_{t+1} = \hat{q}_t$. Nous arrivons alors à la définition de l'algorithme $EXPLORE$ sous la forme d'un problème d'optimisation :

$$EXPLORE(t) = \begin{cases} \text{Maximiser } d(EL_t, \hat{q}) \\ \text{sous la contrainte :} \\ q \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre}; \ell, EL_t) \end{cases}$$

Théorème 3.3 (Exploration)
Soit $\hat{q}_\bullet, \hat{q}_\circ \in \mathcal{C}^{\varepsilon}_{libre}$.
Si $EXPLORE(t) < \varepsilon \quad \wedge \quad \hat{q}_\bullet \notin U(EL_t, \varepsilon)$ *alors* $\mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\circ}} \neq \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\bullet}}$.

La démonstration de ce théorème est faite en utilisant un raisonnement par l'absurde. Pour la comprendre il faut tout d'abord faire l'hypothèse de l'existence d'un chemin $\hat{\varphi}$ *de* $\hat{q}_\circ$ *à* $\hat{q}_\bullet$ *passant à* ε *des* $\mathcal{C}$*-obstacles, autrement dit* $\mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\circ}} = \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\bullet}}$. *Ensuite, il faut noter que dans l'ensemble* EL_t *il existe des balises à une distance inférieure ou égale à* ε *de cette trajectoire, c'est-à-dire de l'image* $\hat{\varphi}([0,1])$*(voir figure 3.6). Nous obtenons alors un sous-ensemble* $EL_t' \subseteq EL_t$ *de balises où chacune de ces balises se touve à une distance inférieure ou égale à* ε *de la trajectoire et par conséquent un ensemble* $U(EL', \varepsilon)$ *de boules centrées dans ces balises. Nous appelons cet ensemble l'ensemble des boules "proches" de la trajectoire. Dans la figure 3.6, nous avons représenté les ensembles de balises* EL_t *et* EL_t' *ainsi que les boules associées. Les boules "proches" du chemin associé aux balises de* EL_t' *sont tracées avec des lignes continues.*

D'un autre côté, si nous savons que $\hat{q}_\bullet \notin U(EL_t, \varepsilon)$*, cela veut dire que* $d(EL_t, \hat{q}_\bullet) > \varepsilon$*. Par conséquent la trajectoire* $\hat{\varphi}$ *entre et sort de chacune des boules "proches" de la trajectoire (dans le cas contraire on aurait* $d(EL_t, \hat{q}_\bullet) \leq \varepsilon$*). Nous avons alors un ensemble fini et discret de "points de sortie" obtenus par l'intersection de la frontière de chaque boule "proche" et de l'image de la trajectoire* $\hat{\varphi}([0,1])$*. Ainsi, nous pouvons obtenir la valeur maximale* $\alpha_t \in [0,1]$ *tel que* $\hat{\varphi}(\alpha_t)$ *est un point de sortie. Evidement ce point* $\hat{\varphi}(\alpha)$ *se trouve à une distance* ε *d'une de balises; notons L cette balise. Nous avons finalement* $Bl(\hat{\varphi}(\alpha_t), \varepsilon) \in \mathcal{C}_{libre}$ *étant donné que* $\hat{\varphi}([0,1]) \in \mathcal{C}^{\varepsilon}_{libre}$ *et il existe alors une trajectoire Manhattan* *d'ordre 1 de L à $\hat{\varphi}(\alpha_t)$ et $d(L, \hat{\varphi}(\alpha_t)) = \varepsilon$ ce qui est en contradiction avec $EXPLORE(t) < \varepsilon$.*

Démonstration par l'absurde :

Prémisses :

$$\begin{aligned} EXPLORE(t) < \varepsilon \quad &\equiv \quad \forall \hat{q} \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre}; \ell, EL_t), \quad d(\hat{q}, EL_t) < \varepsilon \\ \hat{q}_\bullet \notin U(EL_t, \varepsilon) \quad &\equiv \quad d(\hat{q}_\bullet, EL_t) > \varepsilon \end{aligned}$$

Hypothèse :

$$\mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\circ}} = \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\bullet}} \tag{3.10}$$

Nous déduisons de 3.10 que :

$$\exists \hat{\varphi} : I \rightarrow \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\circ}} \quad \textit{tel que} \quad \hat{\varphi}(0) = \hat{q}_\circ, \hat{\varphi}(1) = \hat{q}_\bullet \tag{3.11}$$

Soit $\Lambda = \{\alpha \in I | d(EL_t, \hat{\varphi}(\alpha)) = \varepsilon\}$, nous notons par α_t la valeur maximale de Λ, c'est-à-dire :

$$\alpha_t = \max \alpha \in \Lambda$$

Notons maintenant $L = (x_1, x_2, \ldots, x_n)$ la balise la plus proche de $\hat{\varphi}(\alpha_t) = ({x_1}', {x_2}', \ldots, {x_n}')$. Il est facile de voir que $d(L, \hat{\varphi}(\alpha_t)) = \varepsilon$ car $d(\hat{q}_\bullet, EL_t) > \varepsilon$. (La trajectoire traverse un ou plusieurs fois la boule $Bl(L, \varepsilon)$).

D'autre part nous avons de 3.11 : $Bl(\hat{\varphi}(\alpha_t), \varepsilon) \in \mathcal{C}_{libre}$. Ainsi, il existe un chemin Manhattan $\hat{\gamma}_t$ en $\mathcal{C}_{libre}$ d'ordre 1 de L à $\hat{\varphi}(\alpha_t)$ tel que :

$$\hat{\gamma}_t = ({x_1}' - x_1, {x_2}' - x_2, \ldots, {x_n}' - x_n)$$

De ce fait, $\exists \hat{q}_\alpha = \hat{\varphi}(\alpha) \in \mathcal{P}st_{\mathcal{M}}(\mathcal{C}_{libre}; \ell, EL_t)$ tel que $d(\hat{q}_\alpha, EL_t) = \varepsilon$, ce qui est en contradiction avec nos prémisses. $\square$

La valeur de $EXPLORE(t)$ nous renseigne en fait sur la topologie de l'espace libre. En effet, si à l'itération t on n'a pas placé une balise à moins de $EXPLORE(t)$ de la position but cela veut dire qu'il faudra passer par un “couloir” de l'espace des configurations de diamètre inférieur ou égal à $EXPLORE(t)$ pour atteindre le but.

Nous pouvons conclure du théorème précédent que la difficulté de construire un chemin entre deux configurations $\hat{q}_\circ$,$\hat{q}_\bullet$ appartenant à la même composante connexe de $\mathcal{C}_{libre}$ avec l'algorithme $EXPLORE$ est liée à la valeur ε^* où :

$$\varepsilon^* = max\{\varepsilon \in I\!R | \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\circ}} = \mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\bullet}}\}$$

La difficulté est alors définie comme :

$$\text{difficulté} = \frac{1}{\varepsilon^*}$$

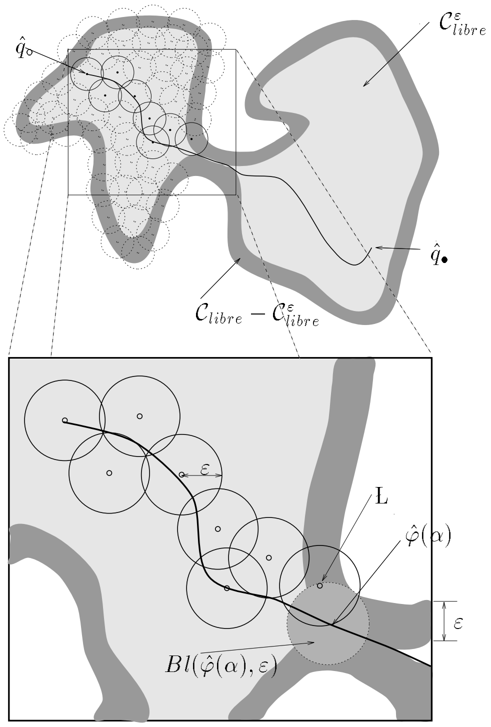


FIG. 3.6 - Pavage de $\mathcal{C}_{libre}$.

Ainsi, pour $\varepsilon \leq \varepsilon^*$, si $EXPLORE(t) \leq \varepsilon$ alors $d(EL_t, \hat{q}_\bullet) \leq \varepsilon$.

Autrement dit, plus ε^* est grand plus $EXPLORE$ trouvera rapidement une solution. Enfin, en utilisant le résultat de la convergence d'$EXPLORE_\infty$ on montre facilement que $\lim_{t\to\infty} EXPLORE(t) = 0$ et donc que l'on peut approcher tout point de $\mathcal{C}^{\varepsilon}_{libre_{\hat{q}_\bullet}}$ en un nombre fini d'itérations.

Théorème 3.4 (Convergence)
Soit X un espace compact, nous avons pour l'application de $EXPLORE$ dans X :

$$\lim_{t\to\infty} EXPLORE(t) = 0 \tag{3.12}$$

Démonstration :

Par définition $\forall t \ EXPLORE(t) \leq EXPLORE_{\infty}(t)$ ce qui d'après le théorème 3.1 nous donne le résultat. $\square$

Chapitre 4

Améliorations de l'algorithme *EXPLORE*

4.1 L'algorithme *SEARCH*

4.1.1 Définition

Dans le chapitre précédent nous avons vu que $\mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ est l'ensemble des chemins Manhattan dans l'espace libre partant de $\hat{q}_\circ$. Ainsi, nous pouvons maintenant définir une fonction $F : \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ) \to I\!R$ prenant pour valeur la distance Euclidienne de l'extrémité d'une trajectoire $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ à une configuration $\hat{q}_\bullet \in \mathcal{C}_{libre}$:

$$F(\hat{\gamma}, \hat{q}_\bullet) = \|\hat{\gamma}(1) - \hat{q}_\bullet\|$$

Nous pouvons donc exprimer le problème de la recherche d'un chemin Manhattan sans collision d'une configuration $\hat{q}_\circ$ à une autre $\hat{q}_\bullet$ comme un problème d'optimisation [6]. Ce problème s'exprime comme :

$$SEARCH(\hat{q}_\circ, \hat{q}_\bullet) = \begin{cases} \text{Minimiser } F(\hat{\gamma}, \hat{q}_\bullet) \\ \text{sous la contrainte :} \\ \hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ) \end{cases}$$

S'il existe un chemin Manhattan $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ reliant la configuration initiale à la configuration finale alors :

$$SEARCH(\hat{q}_\circ, \hat{q}_\bullet) = 0$$

Bien qu'il n'existe pas de forme analytique de F, nous pouvons calculer $F(\hat{\gamma}, \hat{q}_\bullet)$ pour tout $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ et utiliser une méthode d'optimisation pour trouver un optimum (global ou local) à cette fonction. Etant donnée une méthode d'optimisation, on peut alors définir implicitement la pré-image de $\hat{q}_\bullet$ produite par notre planificateur comme :

$$\mathcal{P}_{\mathcal{L}}(\hat{q}_\bullet) = \{\hat{q} \in \mathcal{C}_{libre} | SEARCH(\hat{q}, \hat{q}_\bullet) = 0\}$$

La dimension de cette pré-image dépend à la fois du type d'algorithme d'optimisation utilisé et de la position du but $\hat{q}_\bullet$. Dans le paragraphe suivant nous proposons une amélioration de cette fonction. Elle permet d'agrandir cette pré-image et donc d'augmenter encore l'efficacité de l'algorithme EXPLORE.

4.1.2 La "pré-image Manhattan" d'ordre 1

Soit un point final $\hat{q}_\bullet$ dans l'espace des configurations d'un système à n degrés de liberté. Il est possible de définir une région où cette configuration peut être atteinte avec des déplacements "simples" [48][25], c'est-à-dire une région où $\hat{q}_\bullet$ est "visible". Il existe plusieurs manières de définir les pré-images. Nous définissons un type particulier de pré-image: *la "pré-image Manhattan"* d'ordre 1.

Définition 4.1 (Prédicat MP)
Soit $\hat{q}_a, \hat{q}_b \in \mathcal{C}_{libre}$ deux configurations d'un système à n degrés de liberté, le prédicat *$MP(\hat{q}_a, \hat{q}_b)$ est vrai si et seulement s'il existe un chemin $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; 1, \hat{q}_a)$ tel que $\hat{\gamma}(1) = \hat{q}_b$.*

Définition 4.2 (Pré-image produite par les chemins Manhattan)
Pour une configuration $\hat{q}_a \in \mathcal{C}_{libre}$ la pré-image Manhattan *de $\hat{q}_a$ est définie par :*

$$\mathcal{P}_{\mathcal{M}}(\hat{q}_a) = \{\hat{q} \in \mathcal{C}_{libre} | MP(\hat{q}, \hat{q}_a) = vrai\} \qquad (4.13)$$

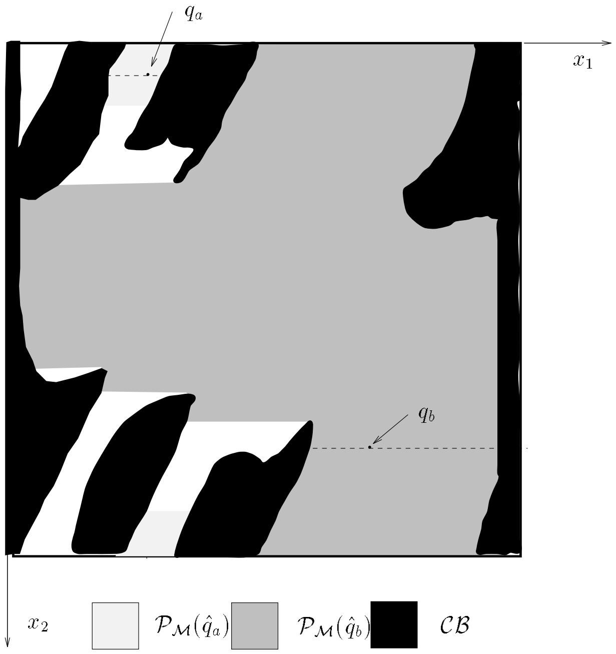


FIG. 4.1 - La pré-image de deux points en $\mathcal{C}_{libre}$.

La figure 4.1 montre la pré-image (Manhattan d'ordre 1) pour deux configurations $\hat{q}_a, \hat{q}_b \in \mathcal{C}_{libre} \subset I\!R^2$. Notons que la pré-image de $\hat{q}_a$ est beaucoup plus petite que celle de $\hat{q}_b$. De là nous pouvons déduire que la planification d'un chemin Manhattan d'une configuration donnée $\hat{q}_\circ \in \mathcal{C}_{libre}$ à $\hat{q}_a$ est plus complexe que celle de $\hat{q}_\circ$ à $\hat{q}_b$.

Evidemment, une trajectoire de $\hat{q}_\circ$ à $\hat{q}_\bullet$ existe s'il existe une trajectoire $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ telle que $d(\hat{\gamma}(1), \mathcal{P}_{\mathcal{M}}(\hat{q}_\bullet)) = 0$. Nous pouvons alors redéfinir la fonction F comme suit :

$$F(\hat{\gamma}, \hat{q}_\bullet) = \begin{cases} 0 & \text{si } \hat{\gamma}(1) \in \mathcal{P}_{\mathcal{M}}(\hat{q}_\bullet) \\ d(\hat{\gamma}(1), \hat{q}_\bullet) & \text{autrement} \end{cases}$$

De cette manière la trajectoire de $\hat{q}_\circ$ à $\hat{q}_\bullet$ serait:

$$\hat{\varphi}(\alpha) = \begin{cases} \hat{\gamma}(2\alpha) & \text{pour} \quad 0 \leq \alpha \leq 1/2 \\ \hat{\gamma}_a(2\alpha - 1) & \text{pour} \quad 1/2 \leq \alpha \leq 1 \end{cases}$$

où $\hat{\gamma}_a$ est obtenue à partir de $\hat{\gamma}(1) = (x_1, x_2, \ldots, x_n)$ et $\hat{q}_\bullet = (x_1^f, x_2^f, \ldots, x_n^f)$, c'est-à-dire :

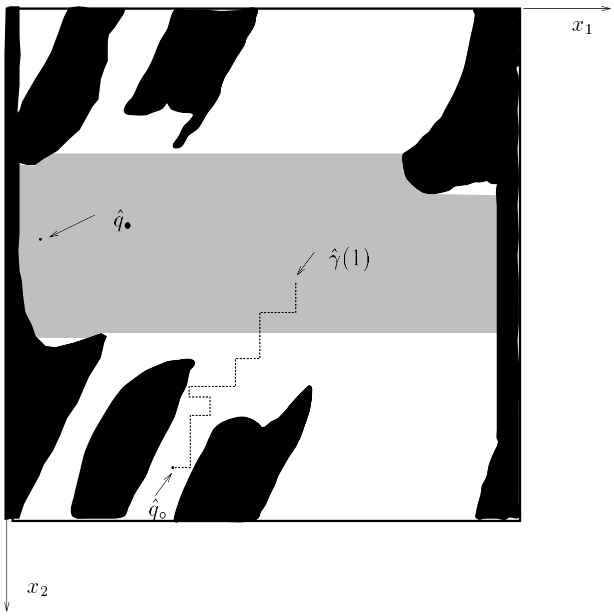


FIG. 4.2 - Une trajectoire où $\hat{\gamma}(1) \in \mathcal{P}_{\mathcal{M}}(\hat{q}_\bullet)$.

$$\hat{\gamma}_a = (x_1^f - x_1, x_2^f - x_2, \ldots, x_n^f - x_n)$$

4.2 Génération efficace de chemins Manhattan en $\mathcal{C}_{libre}$

Si nous voulons appliquer les algorithmes *SEARCH* et *EXPLORE*, nous devons être capables d'engendrer des trajectoires Manhattan d'ordre $k \leq \ell$ dans l'espace libre d'un système à n degrés de liberté. Dans la suite de ce paragraphe nous décrivons une technique pour engendrer de telles trajectoires à partir d'un vecteur $\hat{x} \in I\!R^{n*\ell}$.

4.2.1 L'ensemble propre d'une trajectoire Manhattan

Nous avons vu dans le paragraphe précédent que tout chemin Manhattan $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ d'ordre $k \leq \ell$ pour un robot à n degrés de liberté est codé de la manière suivante :

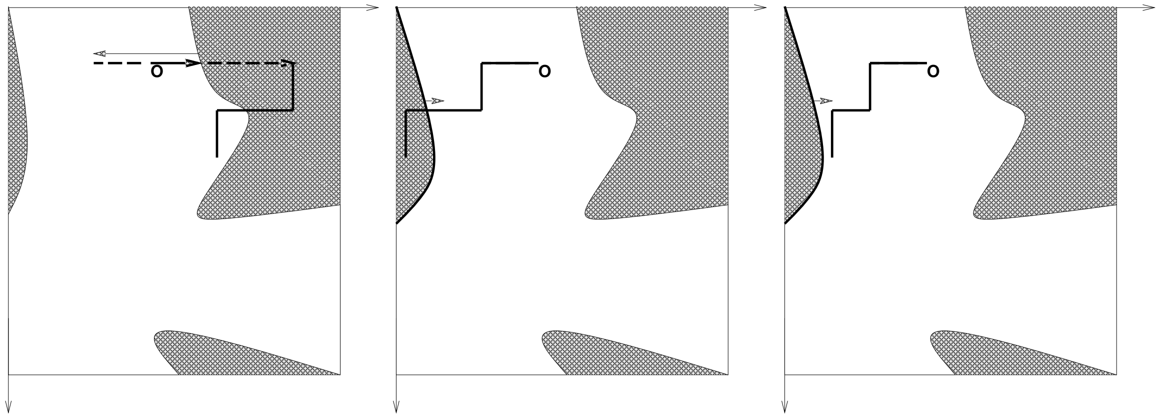
$$\hat{\gamma} = (\Delta_1^1, \Delta_2^1, \ldots, \Delta_n^1, \ldots, \Delta_i^j, \ldots, \Delta_n^k) \tag{4.14}$$

c'est-à-dire que nous avons $k * n$ configurations intermédiaires $\{q_1, q_2, \ldots, q_{k*n}\}$. Nous définissons de cette façon :

Définition 4.3 (Ensemble propre d'un chemin Manhattan)
Soit $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_0)$ un chemin Manhattan d'ordre $k \leq \ell$. On définit l'ensemble propre *de $\hat{\gamma}$, noté $Q(\hat{\gamma}) = \{\hat{q}_0, \hat{q}_1, \ldots, \hat{q}_{k*\ell}\}$, comme l'ensemble des configurations intermédiaires de $\hat{\gamma}$ où :*

$$\begin{aligned} \hat{q}_{m-1} &= (x_1, x_2, \ldots, x_i, \ldots, x_n), \\ \hat{q}_m &= (x_1, x_2, \ldots, x_i + \Delta_i^j, \ldots, x_n) \end{aligned}$$

avec $m = n * (j-1) + i$



a b c

FIG. 4.3 - Le concept de rebondissement.

4.2.2 Le "rebondissement" des trajectoires

Nous pouvons remarquer que toute trajectoire $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ est un vecteur de $\mathbb{R}^{n*\ell}$ mais que le contraire n'est pas forcément vrai. Autrement dit, il n'est pas vrai que tout vecteur en $\mathbb{R}^{n*\ell}$ code une trajectoire en $\mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$.

Par conséquent, $\mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ est un sous-ensemble de $\mathbb{R}^{n*\ell}$ et évidemment l'ensemble $\mathbb{R}^{n*\ell} - \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$ code des trajectoires avec collision. Dans la suite nous proposons une sémantique capable d'engendrer à partir de tout vecteur $\hat{x} \in \mathbb{R}^{n*\ell}$ une trajectoire Manhattan $\hat{\gamma} \in \mathcal{M}(\mathcal{C}_{libre}; \ell, \hat{q}_\circ)$. Pour cela nous considérons que la trajectoire "rebondit" sur les *C-obstacles*. Ainsi, la trajectoire originale de la figure 4.3a se transforme-t-elle en la trajectoire de la figure 4.3b lors de la première collision et en la trajectoire 4.3c lors de la deuxième collision.

Notons $\hat{x} = (\Delta_1^1, \Delta_2^1, \ldots \Delta_n^1, \ldots, \Delta_n^k) \in \mathbb{R}^{n*k}$. La procédure de transformation d'un vecteur $\hat{x}$ en une trajectoire Manhattan est la suivante :

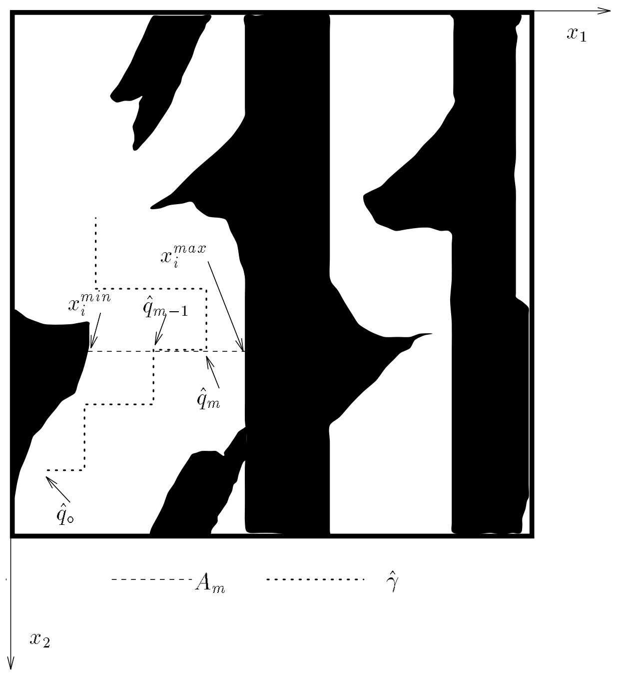


FIG. 4.4 - Le segment A_m calculé à partir d'une configuration $\hat{q}_{m-1}$.

On calcule à partir de $\hat{q}_0 = (x_1^\circ, x_2^\circ, \ldots, x_n^\circ)$ l'intervalle de débattement $A_1 = [x_1^{min}, x_1^{max}]$ parallèle à l'axe x_1. Ainsi, la valeur $x_1^{\circ\prime}$ obtenue après avoir avancé de Δ_1^1 sur l'intervalle A_1 est obtenue en fonction de x_1° et Δ_1^1; $x_1^\circ = rebond(A_1, x_1^\circ, \Delta_1^1)$. On obtient alors $\hat{q}_1 = (x_1^{\circ\prime}, x_2^\circ, \ldots, x_n^\circ)$. On exécute la même procédure pour $\hat{q}_1$ afin d'obtenir $\hat{q}_2 = (x_1^{\circ\prime}, x_2^{\circ\prime}, x_3^\circ \ldots, x_n^\circ)$, et ainsi de suite jusqu'à l'obtention complète de l'ensemble propre de $\hat{\gamma}$.

L'idée générale de la fonction *rebond* est de trouver la configuration $\hat{q}_m = (x_1^m, x_2^m, \ldots, x_n^m)$, atteinte en partant d'une configuration $\hat{q}_{m-1}$ et en se déplaçant d'une distance x_i^j dans l'intervalle A_m (voir figure 4.4). Si au moment de parcourir une distance $|\delta| < |x_i^j|$ on arrive à x_i^{max} ou x_i^{min}, la trajectoire "rebondit" et continue dans le sens inverse. On exécute alors cette même procédure pour le reste de la valeur x_i^j, c'est-à-dire pour $(x_i^j - \delta)$.

Avec cette technique tout vecteur $\hat{x} \in I\!R^{n*\ell}$ code une trajectoire Manhattan de longueur $k \leq \ell$ dans l'espace libre de $\mathcal{C}$. Ainsi, nous pouvons coder toutes les trajectoires Manhattan de longueur $k \leq \ell$ avec $I\!R^{n*\ell}$. D'une manière similaire il est possible avec l'espace $[1, 2, \ldots, t] \times R^{n*\ell}$ de coder toutes les trajectoires de $\mathcal{M}(\mathcal{C}_{libre}; \ell, EL_t)$. Dans ce cas, la valeur $k \in [1, 2, \ldots, t]$ indique la balise L_k de départ et $\hat{x} \in I\!R^{n*\ell}$ la trajectoire Manhattan.

Ainsi, les problèmes d'optimisation posés par les algorithmes $EXPLORE$ et $SEARCH$ peuvent être réécrits comme :

$$EXPLORE(t) = \begin{cases} \text{Maximiser } d(EL_t, \hat{\gamma}(1)) \\ \hat{\gamma} \in [1, 2, \dots, t] \times I\!R^{n*\ell} \end{cases}$$

$$SEARCH(\mathrm{L}_t, \hat{q}_\bullet) = \begin{cases} \text{Minimiser } F(\hat{\gamma}, \hat{q}_\bullet) \\ \hat{\gamma} \in I\!R^{n*\ell} \end{cases}$$

4.3 L'algorithme Fil d'Ariane

Nous obtenons finalement l'Algorithme Fil d'Ariane comme la combinaison des deux problèmes d'optimisation :

Début : $L_1 \leftarrow \hat{q}_\circ, EL \leftarrow \{L_1 = \hat{q}_\circ\}, t \leftarrow 1$

1. Execution de SEARCH : Si $SEARCH(L_t, \hat{q}_\bullet) = 0$ alors un chemin a été trouvé.
2. Execution d'EXPLORE : Placer une nouvelle balise L_{t+1} avec la fonction $EXPLORE(t)$, $EL \leftarrow EL \cup \{L_{t+1}\}$, $t \leftarrow t + 1$, *goto 1.*

Partie II

Application au problème de planification de trajectoires de robots

Chapitre 5

Les Algorithmes Génétiques

Pour résoudre les problèmes d'optimisation impliqués dans l'algorithme Fil d'Ariane, nous utilisons une technique d'optimisation stochastique : les **algorithmes génétiques**. Dans ce chapitre, nous décrivons les principes de ces algorithmes. Après un rappel des problèmes d'optimisation à traiter, nous présentons l'algorithme génétique standard. Nous donnons un exemple illustratif d'application de cet algorithme au problème de la fonction cinématique inverse. La parallélisation des algorithmes génétiques est alors décrite. En particulier, nous introduisons les trois modèles les plus classiques d'algorithmes génétiques parallèles.

L'application des algorithmes génétiques au problème spécifique qui nous intéresse, à savoir la mise en œuvre de l'algorithme Fil d'Ariane, est abordée. Cependant cette implantation ne sera décrite en détail que dans le chapitre 8. Finalement, nous explicitons les raisons qui ont motivé le choix des algorithmes génétiques.

5.1 Rappel des problèmes d'optimisation à traiter

Dans la première partie de cette thèse, nous avons défini l'algorithme Fil d'Ariane comme la combinaison de deux sous-algorithmes : $EXPLORE$ et $SEARCH$. Chacun de ces algorithmes est exprimé comme un problème d'optimisation :

$$EXPLORE(t) = \begin{cases} \text{Maximiser } d(EL_t, \hat{\gamma}(1)) \\ \hat{\gamma} \in [1, 2, \ldots, t] \times I\!R^{n*\ell} \end{cases}$$

$$SEARCH(\hat{q}_\circ, \hat{q}_\bullet) = \begin{cases} \text{Minimiser } F(\hat{\gamma}_{\hat{q}_\circ}, \hat{q}_\bullet) \\ \hat{\gamma}_{\hat{q}_\circ} \in I\!R^{n*\ell} \end{cases}$$

L'espace de recherche de $EXPLORE$ code l'ensemble des trajectoires partant des balises $\{L_1, L_2, \ldots, L_n\}$. L'objectif de la fonction d'évaluation de $EXPLORE$ est de trouver une trajectoire $\hat{\gamma}$ dont l'extrémité $\hat{\gamma}(1)$ soit la plus éloignée possible des balises précédemment posées.

L'espace de recherche de $SEARCH$ représente l'ensemble des trajectoires partant de la dernière balise posée par $EXPLORE$. L'objectif de la fonction d'évaluation de $SEARCH$ est de trouver une trajectoire $\hat{\gamma}$ telle que son extrémité $\hat{\gamma}(1)$ se trouve dans la pré-image Manhattan de la configuration finale $\hat{q}_\bullet$.

Pour résoudre ces deux problèmes d'optimisation, nous avons utilisé les algorithmes génétiques. Nous décrivons dans la suite de ce chapitre cette technique d'optimisation.

5.2 Introduction aux algorithmes génétiques

Les algorithmes génétiques sont une technique de recherche stochastique introduite par Holland [35] dans les débuts des années 70. Ils sont inspirés par l'évolution biologique des espèces.

Avec le développement des architectures parallèles, ils sont en train de connaître un certain succès : un nombre croissant de travaux portent sur l'application des algorithmes génétiques pour les problèmes d'optimisation combinatoire, en intelligence artificielle, dans le domaine du connexionisme (voir [44] [37] [33]) et pour la robotique([4] [45] [26] [27] [58]).

Le but des algorithmes génétiques est de trouver l'optimum d'une certaine fonction F sur un espace de recherche S. Par exemple, l'espace de recherche S peut être

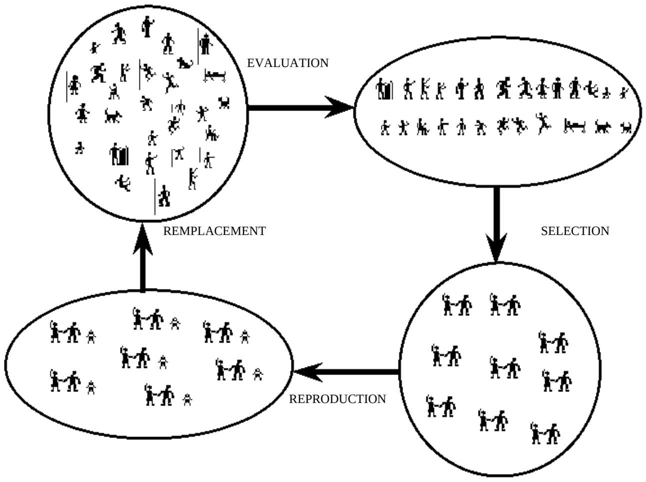


FIG. 5.1 - Etapes d'un algorithme génétique.

2^N. Un point de S est alors décrit par un vecteur de N bits et F est une fonction permettant de calculer une valeur réelle pour chacun de ces 2^N vecteurs.

Dans une phase d'initialisation, un ensemble de points de S (appelé "population d'individus") est tiré aléatoirement (le "génotype" de chaque individu est un vecteur de N bits). L'algorithme génétique itère alors sur les 4 étapes suivantes jusqu'à ce qu'un optimum satisfaisant soit atteint (voir figure 5.1) :

1. **Evaluation** : La fonction F est calculée pour chaque individu, de manière à classer toute population du moins bon jusqu'au meilleur.
2. **Sélection** : Des paires d'individus sont sélectionnées, les individus les meilleurs ayant plus de chances d'être sélectionnés que les autres (un individu peut être éventuellement sélectionné plusieurs fois).
3. **Reproduction** : De nouveaux individus (appelés "descendants") sont produits à partir de ces paires.
4. **Remplacement** : Une nouvelle population est engendrée en remplaçant certains des individus de la vieille population par des jeunes venant d'être créés.

La reproduction est effectuée en utilisant des "opérateurs génétiques". Il existe une grande variété de tels opérateurs [20] mais les deux plus communs sont la "mutation" et le "cross-over" ou combinaison.

L'opérateur de mutation consiste à tirer au hasard certains points de la chaîne de bits du génotype et à basculer la valeur binaire de ces "gènes" (voir figure 5.2).

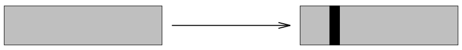


FIG. 5.2 - Mutation d'un individu.

L'opérateur de *cross-over* sélectionne au hasard un point de coupure parmi N sites possibles dans le génotype binaire et échange les parties pré-coupure et post-coupure des deux vecteurs parents (voir figure 5.3).

Les algorithmes génétiques ont de nombreuses applications et font preuve de capacités d'optimisation bonnes comparés à d'autres techniques, notament, quand la taille de l'espace de recherche S est très grand et que la fonction F est relativement irrégulière (voir[32][31]).

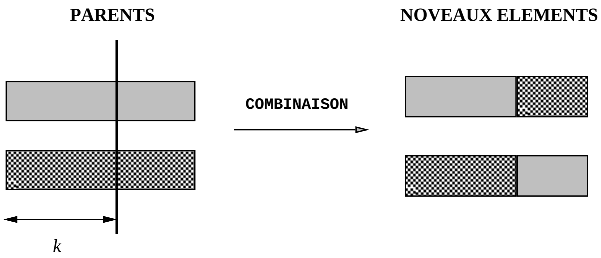


FIG. 5.3 - Combinaison des individus.

Au delà de leur intérêt scientifique en tant que modèle de l'évolution, les algorithmes génétiques ont deux intérêts techniques principaux :

1. Ils sont très "robustes" et peuvent traiter une très grande variété de problèmes d'optimisation.
2. Ils sont très faciles à paralléliser et les accélérations obtenues par ce procédé sont considérables (voir par exemple [83] et le paragraphe 5.3.4).

5.2.1 Exemple : Le problème de la fonction cinématique inverse

Afin d'introduire les algorithmes génétiques et leur application dans les algorithmes $EXPLORE$ et $SEARCH$, nous présentons dans ce paragraphe la solution à un problème typique de robotique : l'inversion de coordonnées [19]. Etant donné un bras manipulateur, le problème consiste à trouver une configuration du bras telle que l'extrémité de la dernière articulation soit à une position donnée $X = (x, y)$ de l'environnement. On impose en plus à cette configuration d'être libre de collision. Considérons un bras planaire à deux degrés de liberté. Soit $\Theta = (\theta_0, \theta_1)$ une confi-

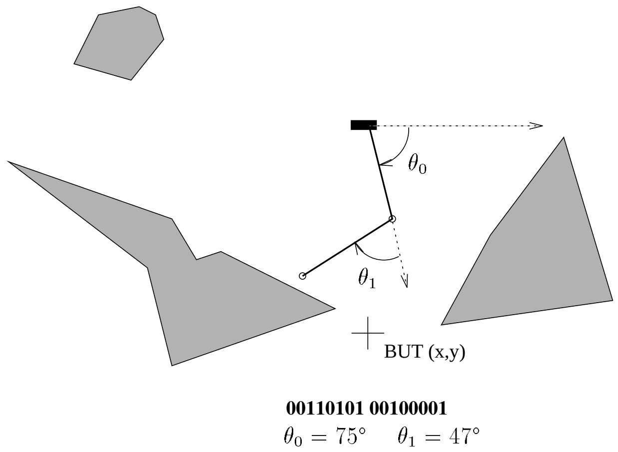


FIG. 5.4 - Une configuration du bras manipulateur et son codage.

guration quelconque du bras; alors le problème d'inversion de coordonnées peut être exprimé comme un problème de minimisation :

$$\min_{\Theta} f(\Theta) : f(\Theta) = \begin{cases} \|X_\Theta - X\| & \text{si } \Theta \in \mathcal{C}_{libre} \\ +\infty & \text{Collision avec un obstacle} \end{cases} \tag{5.15}$$

Où X_Θ dénote la position de l'extrémité de la dernière articulation étant donnée la configuration Θ (voir figure 5.4).

On constate aisément que la résolution analytique de cette expression n'est pas évidente lorsqu'il y a des obstacles. Néanmoins, le calcul direct de f pour une configuration donnée Θ peut être facilement obtenu.

Codage du problème

Pour coder une configuration Θ, nous utilisons des vecteurs de 16 bits. Les 8 premiers bits représentent θ_0 et les 8 restants θ_1. De cette manière, nous discrétisons l'espace des configurations ($[0, 2\pi[\times[0, 2\pi[$) en couples d'entiers (i_0, i_1) où i_0,$i_1 \in 0, \ldots, 255$. Un individu quelconque représente alors une configuration. Par exemple l'individu A_i = 00000001 00000010 ou $i_0 = 1$ et $i_1 = 2$ représente la configuration $\Theta = (2\pi/256, 4\pi/256)$. Cette configuration a une valeur f_i donnée par la fonction $f(g(A_i))$ où $g(A_i)$ est la transformation de (i_0, i_1) en (θ_0, θ_1).

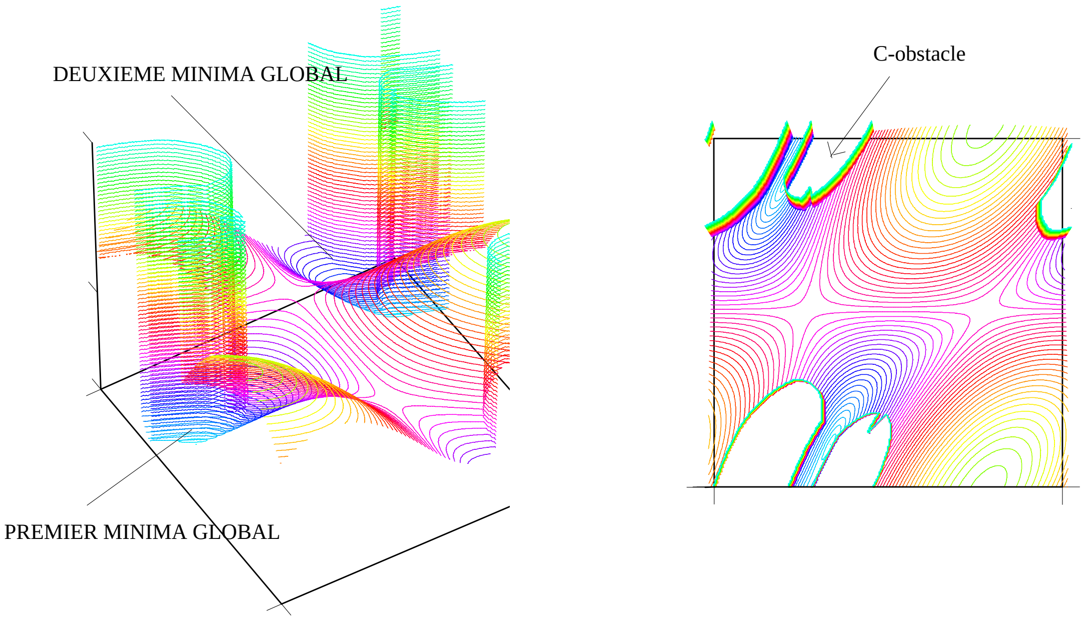


FIG. 5.5 - Graphique de la fonction d'inversion de coordonnées.

Résolution du problème

Nous considérons pour notre exemple l'environnement et le bras planaire de la figure 5.4. Etant donné que nous utilisons 8 bits pour représenter chacun des angles du robot, l'espace de recherche de l'algorithme génétique est

$$S = \{0,1\}^{16} = \{0,1\}^8 \times \{0,1\}^8$$

Pour notre environnement, la fonction f est représentée graphiquement dans la figure 5.5. La résolution du problème consiste à exécuter l'algorithme génétique standard décrit précédemment et montré en pseudo "c" ci-dessous :

AG()
begin
$t = 0$;
Initialiser(*Population*)
Evaluer(*Population*)
while ($\neg$ condition_arrêt)
  $t = t + 1$;
  /*–Créer les couples–*/

FIG. 5.6 - Distribution de la population initiale.

Couples = Sélection($Population$);
/*–Créer les descendants–*/
$Descendants$ = Reproduction($Couples$);
/*–Substituer la population actuelle par les meilleurs descendants–*/
$Population$ = Remplacement($Descendants$);
/*–Evaluer la nouvelle population–*/
Evaluer($Population$);
end

Une fois l'algorithme lancé, l'évolution de la population converge très rapidement vers les deux configurations qui permettent de parvenir au point de référence donné dans l'espace de travail.

Dans la figure 5.6, nous avons représenté la population initiale répartie dans l'espace des configurations. Bien évidemment, il y a des configurations valides (sans collision) et des configurations invalides. La figure 5.7 montre la dynamique de l'évolution de la population. Les lignes indiquent la substitution d'un élément après chaque remplacement. Autrement dit, l'élément A_i a été remplacé par un élément meilleur A_i^1 puis par A_i^2, etc. Par exemple, dans la figure 5.7, nous montrons "l'arbre généalogique de A_5; $(A_5, A_5^1, A_5^2, A_5^3)$. L'élément A_5^3 est finalement une des solutions au problème. Par conséquent, l'élément A_5^3 n'est plus remplacé par sa descendance. Un deuxième point intéressant à remarquer est la disparition dès les premières générations des individus très mauvais qui codent des configurations du bras en collision avec les obstacles [2].

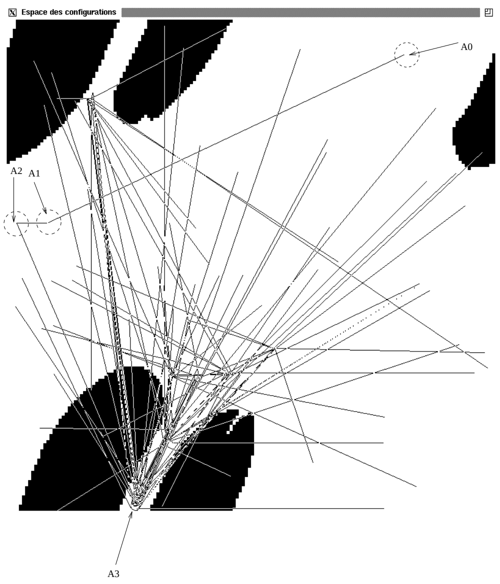


FIG. 5.7 - Convergence de la population.

5.3 La parallélisation des algorithmes génétiques

Pour certains problèmes d'optimisation, le coût d'exécution des algorithmes génétiques est un obstacle majeur à leur développement. L'apparition des architectures massivement parallèles ouvre de nouveaux horizons pour ces algorithmes : un nombre croissant de travaux utilisant des versions parallèles de ces algorithmes ont été réalisés pour diverses applications.

Deux modèles parallèles pour les algorithmes génétiques sont cités dans la littérature : le **modèle parallèle standard** et le **modèle à décomposition**. Nous décrivons ces deux modèles et nous en proposons un troisième de granularité plus fine, le **modèle massivement parallèle** [1].

5.3.1 Modèle parallèle standard

Ce modèle consiste à utiliser l'algorithme standard en effectuant en parallèle les étapes d'évaluation, de sélection et de reproduction [68]. L'étape de sélection nécessite un graphe d'individus complet, toute paire d'individus de la population étant potentiellement candidate (voir figure 5.8). Un processeur maître effectue la sélection des paires d'individus et les diffuse aux processeurs esclaves. Une fois qu'un

[1] Ce modèle a été conçu en collaboration avec le Laboratoire de Génie Informatique (LGI) de l'IMAG

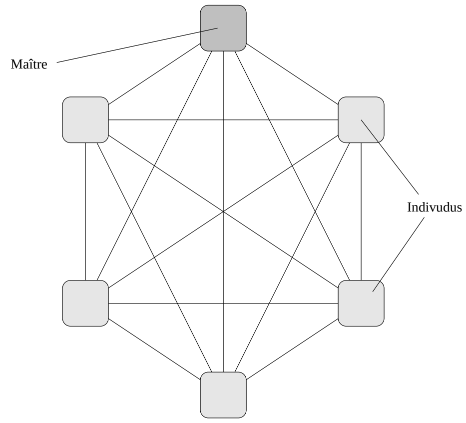


FIG. 5.8 - Modèle parallèle standard.

processeur esclave reçoit une paire d'individus, il effectue sa reproduction, évalue les individus produits, et les réenvoit au processeur maître. Le processeur maître effectue le remplacement de la population courante et relance le même processus à partir de la nouvelle population.

Le modèle parallèle standard n'est pas souple dans le sens où le coût de communication croît exponentiellement en fonction de la taille de la population; c'est pourquoi l'étape de sélection est effectuée séquentiellement. Cette approche est donc mal adaptée aux architectures parallèles à mémoire distribuée, pour lesquelles un faible coût de communication est d'une importance fondamentale pour garantir de bonnes performances.

5.3.2 Modèle à décomposition

Ce modèle consiste à diviser la population en sous-populations de même taille. Chaque processeur exécute l'algorithme génétique standard sur la sous-population qui lui est affectée [64][84]. Un échange d'individus (envois et réception) entre les sous-populations se fait régulièrement. Les meilleurs individus reçus vont ainsi remplacer les plus mauvais de la population locale. Le voisinage d'un processeur, la fréquence des échanges et le nombre d'individus échangés sont des paramètres de l'algorithme (voir figure 5.9).

Dans ce modèle, le parallélisme inhérent aux algorithmes génétiques n'est pas totalement exploité, le traitement d'une sous-population n'étant pas parallélisé. Cette approche est cependant intéressante dans le cas où le nombre de processeurs disponibles est plus petit que la taille de la population désirée. Le grain du parallélisme de ce modèle est moyen.

Il est important de noter que des études effectuées ont montré que la division

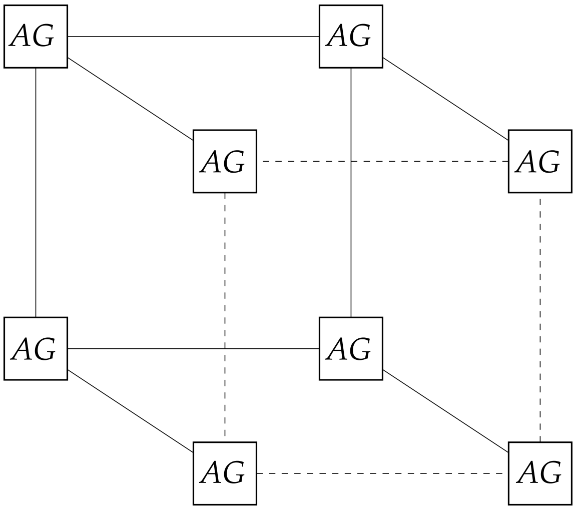


FIG. 5.9 - Modèle parallèle à décomposition.

de la population en sous-populations n'entraîne de dégradation ni de la qualité du résultat ni de l'efficacité de l'algorithme [9].

5.3.3 Modèle massivement parallèle (PGA)

Considérant les architectures massivement parallèles où le nombre de processeurs peut être modulé suivant la taille de la population désirée, nous avons choisi un modèle de granularité fine. Dans cette architecture la population est placée sur un graphe connexe non complètement connecté, un individu par nœud.

Trouver un équilibre entre l'exploration de l'espace de recherche et l'exploitation des meilleurs solutions est un facteur déterminant pour les performances des algorithmes génétiques. L'exploitation avec des bonnes solutions peut aboutir à une convergence prématurée. La sélection dans notre modèle est faite localement dans le voisinage de chaque individu [80]. En conséquence, il existe une pression de sélection moins importante et une tendance vers une plus grande exploration de l'espace de recherche.

Le choix du voisinage est un paramètre de l'algorithme. Pour éviter le coût et la complexité des algorithmes de routage dans les architectures parallèles à mémoire distribuée, un choix judicieux peut être de restreindre le voisinage aux individus directement connectés.

L'algorithme génétique parallèle **PGA** proposé est alors le suivant :

- Engendrer en **parallèle** une population d'individus aléatoires (un individu par nœud).

- **Evaluation :** évaluer **en parallèle** chaque individu.
- Tant que (nombre_de_générations $\leq$ nombre_de_générations_max) Faire
  - **Sélection :** recevoir **en parallèle** les individus voisins.
  - **Reproduction :** chaque individu se reproduit **en parallèle** avec ses voisins.
  - **Evaluation :** évaluer **en parallèle** chaque individu produit.
  - **Remplacement :** effectuer **en parallèle** le remplacement de l'individu local.

Mise en œuvre

Nous avons participé au développement et à l'implantation sur des machines massivement parallèles à base de Transputers de l'algorithme génétique massivement parallèle PGA, voir [83]. Cet algorithme a fait preuve de bonnes performances et surtout d'une accélération considérable de la découverte d'une bonne solution comparée à la version séquentielle. Dans la suite, nous décrivons la mise en œuvre de cet algorithme sur la machine SuperNode : une architecture reconfigurable à base de transputers.

La population est placée sur un tore de processeurs. Etant donné les quatre liens du transputer, chaque individu possède quatre voisins physiques. Le voisinage "naturel" d'un individu est donc composé de ces quatre voisins physiques. C'est parmi ces quatre voisins que se fait la sélection.

A chaque génération, on ne recherche pas la meilleure solution présente dans le réseau : le coût d'une telle recherche serait en effet trop important. Les seules solutions considérées sont celles qui passent à travers le processeur "root" (voir figure 5.10) utilisé pour fermer le tore tout en préservant la possibilité de communiquer avec l'extérieur. A cette fin, un processus espion est donc placé sur ce processeur. A chaque amélioration de la fonction coût, il transmet la nouvelle solution au système hôte chargé de la communication.

Complexité de l'algorithme

Dans ce paragraphe nous abordons la complexité de l'algorithme parallèle proposé. Les notations suivantes sont utilisées :

- n taille de la population,
- s taille du voisinage,

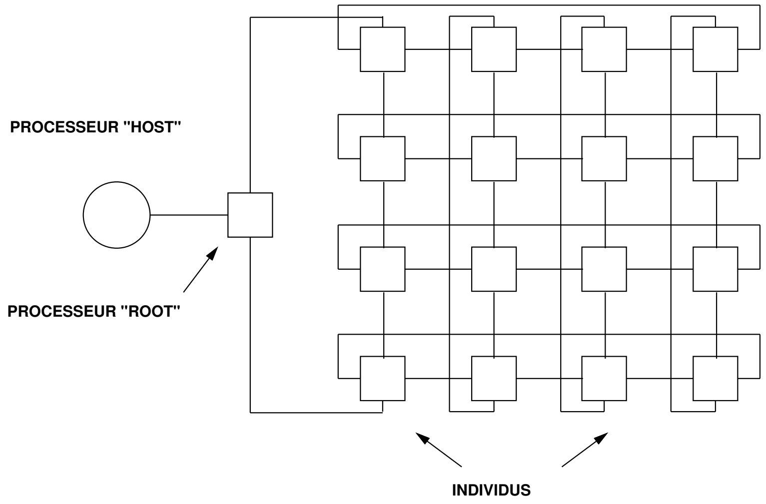


FIG. 5.10 - Un tore de 16 processeurs.

– t taille d'un vecteur représentant une solution du problème.

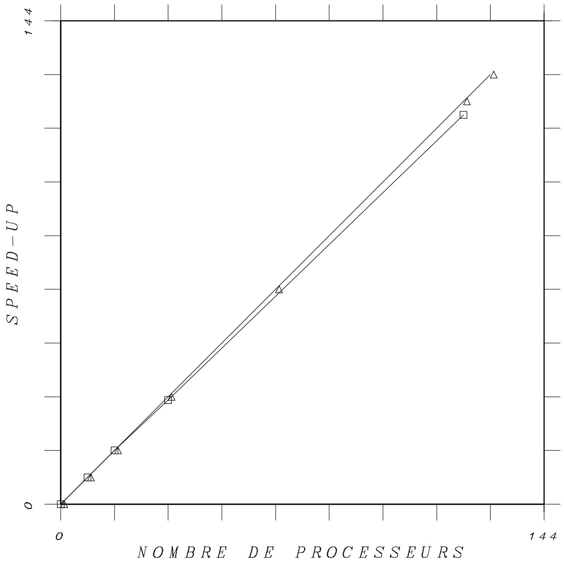
Commençons par calculer la complexité de l'algorithme génétique séquentiel standard. Ceci nécessite le calcul de la complexité des différentes étapes de l'algorithme (sélection, cross-over, mutation,...). L'étape d'évaluation n'est pas considérée puisqu'elle dépend du problème traité mais elle demeure fondamentale.

La complexité de l'étape de sélection est de $O(n^2)$. L'opérateur de *cross-over* nécessite $O(t)$ opérations, ce qui donne une complexité de $O(n \cdot t)$ pour toute la population. De même l'opérateur de mutation, nécessite $O(n \cdot t)$ opérations. La complexité de l'étape de remplacement est de $O(n \cdot log(n))$ [1]. Soit, pour l'algorithme complet, une complexité de $O(n^2 + n \cdot t)$.

Pour l'algorithme génétique parallèle, la complexité de l'étape de sélection est de $O(s)$. L'étape de reproduction a une complexité en $O(s \cdot t)$ aussi bien pour l'opérateur de *cross-over* que pour la mutation. La complexité de l'étape de remplacement est de $O(s)$. La table 5.1 résume la complexité de l'algorithme génétique séquentiel standard et de l'algorithme parallèle PGA.

opération	**AG séquentiel**	**AG Parallèle**
Sélection	$O(n \cdot n)$	$O(s)$
Cross-over	$O(n \cdot t)$	$O(s \cdot t)$
Mutation	$O(n \cdot t)$	$O(s \cdot t)$
Remplacement	$O(n \cdot log(n))$	$O(s)$
Total	$O(n \cdot n + n \cdot t)$	$O(s \cdot t)$

TAB. 5.1 - Complexité des algorithmes.



△ accélération idéal, □ nos résultats

FIG. 5.11 - Accélération de l'algorithme génétique parallèle.

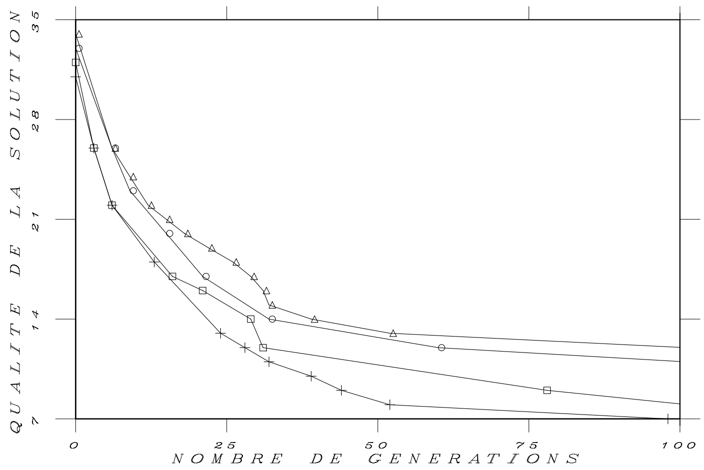
5.3.4 Evaluation des performances de l'algorithme

L'algorithme génétique décrit précédemment a été utilisé pour traiter plusieurs problèmes d'optimisation. Un exemple est le problème du placement d'un graphe de processus sur un graphe de processeurs [80]. Ce problème a été résolu avec succès en utilisant l'algorithme génétique massivement parallèle. Ce travail est présenté dans la thèse de E.G.Talbi [83] avec qui nous avons collaboré. Dans la suite, nous présentons quelques mesures de performances effectuées par E.G. Talbi. Même si ces performances dépendent en partie du problème d'optimisation posé, elles illustrent bien la puissance de l'algorithme PGA.

Variation du nombre de processeurs

Le but de cette évaluation est de mesurer le facteur d'accélération de l'algorithme génétique parallèle PGA (pour une population de taille donnée) sur des tores de processeurs de différentes tailles.

L'accélération S est défini par l'équation suivante : $S = T_s/T_p$ où T_s est le temps d'exécution de l'algorithme sur un seul processeur et T_p correspond au temps d'exécution sur p processeurs. La figure 5.11 montre les résultats obtenus.



Taille de la population : $\triangle = 9$, $\circ = 16$, $\square = 32$, $+ = 64$

FIG. 5.12 - Qualité de la solution en fonction de la taille de la population.

L'algorithme possède une accélération presque idéal. Ceci s'explique par les faits suivants :

- le coût de communication entre processus est relativement petit comparé au coût d'exécution des processus,
- le coût de communication de l'algorithme est indépendant de la taille de l'architecture puisque les communications sont purement locales, limitées au quatre voisins physiques.

Variation de la taille de la population

Le but de cette évaluation est de mesurer l'évolution de la qualité de la solution obtenue pour des populations de taille différentes. La figure 5.12 montre les résultats obtenus en fonction de la taille de la population (le problème traité dans cet exemple est un problème de minimisation).

Comme prévu, plus la taille de la population est grande, meilleure est la qualité du résultat. Ceci montre l'importance de la flexibilité du modèle choisi.

Pour des populations de petites tailles, il est possible qu'une convergence prématurée se produise et que dans ce cas la solution optimale ne soit jamais trouvée.

La figure 5.12 montre aussi que la qualité de la solution s'améliore plus rapidement dans les premières générations que dans les suivantes. Ainsi, une solution de qualité satisfaisante peut être obtenue très rapidement sans laisser l'algorithme se dérouler jusqu'à son terme.

5.4 Utilisation de l'algorithme PGA dans l'algorithme Fil d'Ariane

Les problèmes d'optimisation entrant en jeu dans l'algorithme Fil d'Ariane sont traités par l'algorithme PGA. Plusieurs implantations de l'algorithme PGA ont été développées pour la planification de trajectoires d'un bras manipulateur [17]. Cependant, nous présenterons uniquement dans cette thèse la version finale. Cette présentation sera faite en détail dans le chapitre 8 en même temps que la description de la parallélisation de $SEARCH$, $EXPLORE$ et de la fonction d'évaluation. En effet la parallélisation de l'algorithme Fil d'Ariane et de ses quatre composantes (PGA, $EXPLORE$, $SEARCH$,fonction d'évaluation) est un tout qui se doit d'être présenté de façon unitaire.

5.5 Pourquoi avoir choisi les algorithmes génétiques?

La première qualité des algorithmes génétiques et non la moindre, est leur simplicité. Un algorithme génétique complet (hormis la fonction d'évaluation) ne représente que quelques centaines de lignes de code. C'est évidement un avantage déterminant pour le développement d'applications.

Leur deuxième qualité est la généricité. Il suffit, en théorie, de changer la fonction d'évaluation pour pouvoir appliquer le même algorithme génétique à différents problèmes d'optimisation. En pratique, évidemment, cette affirmation doit être nuancée et il peut arriver que la fonction à optimiser entraîne de légères modifications ou adaptations du code de l'algorithme génétique lui même. Le peu de modifications à faire, et la simplicité des algorithmes génétiques fait qu'il est très facile de modifier un algorithme génétique développé pour un problème d'optimisation pour l'utiliser pour un autre.

Leur troisième qualité est la facilité et l'efficacité de leur parallélisation. Ces propriétés ont été clairement établies au paragraphe précédent. Cette qualité est une des grandes différences qui sépare les algorithmes génétiques de leurs concurrents les plus proches, les algorithmes de type "recuit simulé".

En effet, la famille des algorithmes "recuit simulé" est extrêmement difficile à paralléliser et cette parallélisation conduit souvent à des performances décevantes. Enfin, et surtout, les algorithmes génétiques sont performants pour traiter des problèmes d'optimisation où l'espace de recherche est très grand et où l'ensemble des solutions satisfaisantes est important. Ils ne sont pas du tout adaptés pour trouver une solution unique optimale. Ils sont, par contre, très performants pour découvrir rapidement une solution acceptable (sachant qu'il y en a beaucoup) parmi un nombre colossal de solutions potentielles. Les problèmes d'optimisation posés par l'algorithme Fil d'Ariane sont exactement de ce type. L'espace de recherche (ensemble de trajectoires) est très grand. Les solutions acceptables recherchées (les trajectoires sans collision) sont très nombreuses.

En résumé, les raisons pour lesquelles nous avons choisi d'utiliser les algorithmes génétiques sont les suivantes :

- simplicité de programmation,
- généricité
- simplicité et efficacité de la parallélisation,
- adaptation au type de problème d'optimisation posé par l'algorithme Fil d'Ariane (très grand espace de recherche et grand nombre de solutions).

Chapitre 6

Planification de trajectoires pour un robot mobile holonome

Nous avons utilisé l'algorithme Fil d'Ariane pour planifier les trajectoires d'un robot mobile holonome. Rappelons que dans ce problème il s'agit de trouver un chemin sans collision pour un véhicule pouvant tourner sur place. Dans la maquette réalisée, l'utilisateur peut modifier les positions des obstacles au cours du déplacement du véhicule. Notre objectif est alors de replanifier immédiatement une nouvelle trajectoire vers le but sans interrompre le mouvement du robot. L'objectif visé est de montrer la rapidité du planificateur proposé et la possibilité de construire un planificateur global réactif.

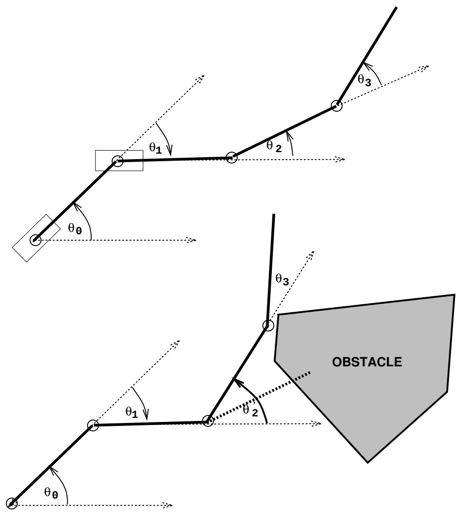


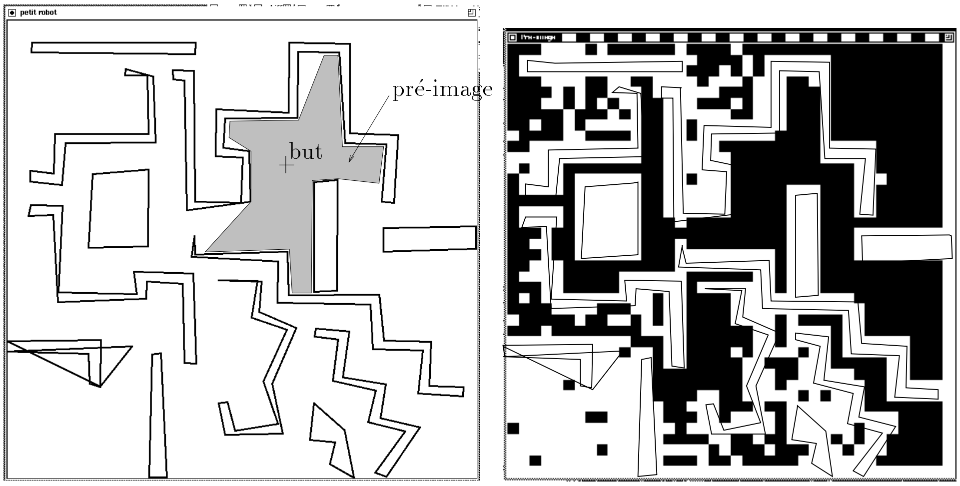
FIG. 6.1 - Codage des trajectoires pour le robot holonome et le rebondissement.

6.1 Description de l'implantation et résultats expérimentaux

6.1.1 Implantation de l'algorithme $SEARCH$

Dans cette application, nous utilisons une version modifiée des chemins Manhattan précédemment décrits. Nous considérons un sous-ensemble discret de tous les chemins partant d'une configuration donnée $\hat{q}_\circ$ et d'une longueur inférieure ou égale à d. Un chemin $\hat{\gamma}$ est codé grâce à k entiers positifs $\{\theta_i\}_{i=0,k-1}$ qui codent l'angle entre deux segments linéaires de longueur égale à $\frac{d}{k}$. Dans un espace complètement libre, ce type de trajectoires est seulement un ensemble de segments connectés de longueur $\frac{d}{k}$. Notons $\hat{q}_i$ le point initial de chacun des segments. Selon notre définition le robot tourne de θ_i au point $\hat{q}_i$; ensuite, il exécute un mouvement en ligne droite vers $\hat{q}_{i+1}$.

Dans un espace encombré d'obstacles, ce codage peut aussi représenter une trajectoire sans collision à condition d'utiliser la notion de “rebond” . Lorsqu'un segment $\{\hat{q}_i, \hat{q}_{i+1}\}$ entre en collision avec un obstacle, la valeur de θ_i est incrémentée ou décrémentée jusqu'à trouver une valeur $\theta_i{}'$ pour laquelle cette collision disparaît. On dit que la trajectoire “rebondit” sur l'obstacle (voir figure 6.1). En utilisant cette technique, un vecteur de k entiers représente une trajectoire d'ordre k et de longueur d. On peut remarquer que cette technique de codage permet une recherche dans l'espace des chemins réalisables formés d'une séquence de rotations et de déplacements de longueur finis.



a b

FIG. 6.2 - La pré-image des mouvements "directs" au but et la pré-image $\mathcal{P}_{\mathcal{L}}$.

Pour tout chemin $\hat{\gamma}$ et donc pour tout vecteur de $\mathbb{N}^k$, nous définissons la fonction $F(\hat{\gamma}, \hat{q}_\bullet)$ comme suit : $F(\hat{\gamma}, \hat{q}_\bullet) = 0$ s'il existe un mouvement "direct" d'une des configurations $\{\hat{q}_i\}_{i=0,k-1}$ au but et $F(\hat{\gamma}, \hat{q}_\bullet) = \min_{i=0,k-1} ||\hat{q}_i - \hat{q}_\bullet||$ sinon. Bien qu'il n'existe pas de forme analytique pour $F(\hat{\gamma}, \hat{q}_\bullet)$, un algorithme génétique peut être utilisé pour minimiser cette fonction de $\mathbb{N}^k$ dans $\mathbb{R}^+$

L'ensemble des configurations pour lesquelles un mouvement direct au but est possible définit implicitement une pré-image (voir figure 6.2a). Une autre pré-image, de taille plus grande, est définie implicitement par la fonction $F(\hat{\gamma}, \hat{q}_\bullet)$ et les algorithmes génétiques. Cette pré-image est définie comme l'ensemble des configurations $\mathcal{P}_{\mathcal{L}}$ où pour tout $\hat{q} \in \mathcal{P}_{\mathcal{L}}$ l'algorithme génétique est capable de trouver un chemin $\hat{\gamma}$ ("rotation" et "déplacement") partant de $\hat{q}$ et tel que $F(\hat{\gamma}, \hat{q}_\bullet) = 0$.

A titre d'exemple, et pour montrer que cette pré-image peut couvrir une partie non négligeable de l'espace libre, nous présentons dans la figure 6.2b la pré-image engendrée par $SEARCH$ en utilisant des trajectoires d'ordre 10 avec une longueur de segment égale à trois fois la longueur du robot. La région noire représente toutes les configurations de départ pour lesquelles une trajectoire a été trouvée vers la configuration finale montrée dans la figure 6.2a. Cette région correspond donc à la pré-image $\mathcal{P}_{\mathcal{L}}$ précédemment définie. On peut remarquer en particulier que les obstacles n'appartiennent pas à cette pré-image.

Ainsi, partant d'une configuration initiale, il suffit de placer une balise dans cette pré-image pour trouver une trajectoire. La dimension de la pré-image est une bonne indication du nombre de balises que devra déposer l'algorithme $EXPLORE$

pour trouver une solution. Plus est grande la pré-image, plus les chances d'y placer rapidement une balise avec $EXPLORE$ est élevée.

Dans l'implantation de $SEARCH$, nous utilisons des trajectoires formées de six rotations et six déplacements. Les déplacements représentent toujours trois fois la longueur du véhicule. Ces paramètres sont définis expérimentalement. Ainsi, une trajectoire pour l'algorithme $SEARCH$ est représentée de la manière suivante :

$$(\theta_1, \ldots, \theta_i, \ldots, \theta_6)$$

où chacun des θ_i est codé par un vecteur S_i de 7 bits, c'est-à-dire 128 valeurs possibles. Un chemin, comme ceux montrés dans la figure 6.3, est trouvé au bout de 7 générations en moyenne de l'algorithme génétique pour une population de 25 individus et dans un temps inférieur à 0.5 secondes.

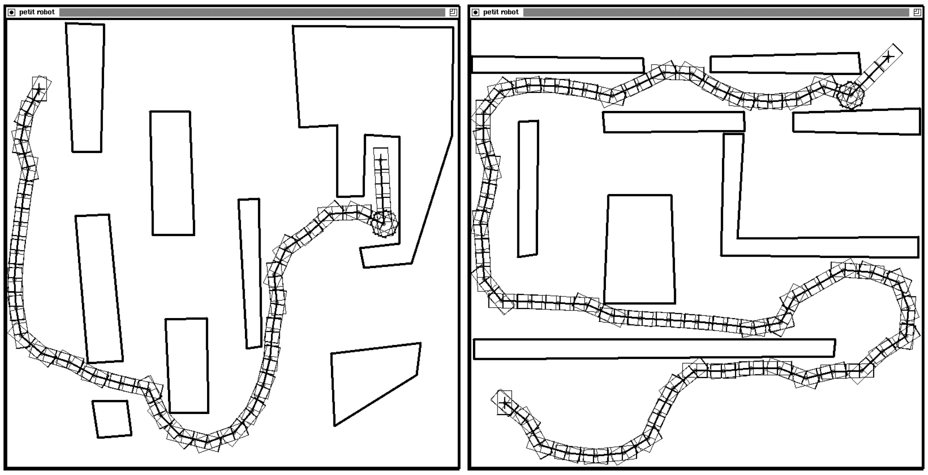


FIG. 6.3 - Deux chemins trouvés par $SEARCH$.

Lorsque le robot mobile est en train d'exécuter une trajectoire, l'utilisateur peut, en utilisant le simulateur, changer la position des obstacles. Le planificateur réagit alors immédiatement et recalcule une nouvelle trajectoire. Par exemple, si la porte représentée dans la figure 6.4 est fermée avant que le robot ne la franchisse, le planificateur est capable de reconstruire instantanément une nouvelle trajectoire.

6.1.2 Implantation de l'algorithme $EXPLORE$

L'espace de recherche de $EXPLORE$ est directement inspiré de l'espace des trajectoires précédemment décrit pour $SEARCH$. Nous utilisons aussi un ensemble de six entiers pour représenter le chemin et nous appliquons la même technique de rebondissement pour engendrer des chemins dans l'espace libre. Nous ajoutons un

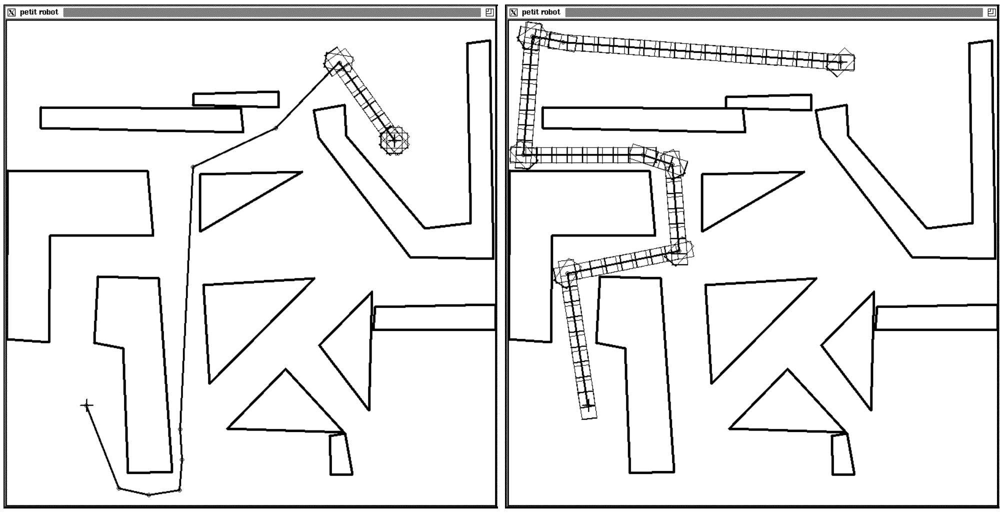


FIG. 6.4 - Planification de trajectoires pour le robot holonome dans un environnement dynamique.

nouvel entier I_t à cet ensemble pour coder la position de départ du chemin. La valeur de I_t code l'indice de la balise de laquelle partira le chemin.

Un algorithme génétique est utilisé pour maximiser la fonction $EXPLORE(t)$. I_t et les θ_i sont codés par un vecteur S_i de 7 bits. Une trajectoire pour l'algorithme $EXPLORE$ est représentée de la manière suivante :

$$(I_t, \theta_1, \ldots, \theta_i, \ldots, \theta_6)$$

Par conséquent, nous définissons implicitement une post-image engendrée par l'ensemble des balises et l'ensemble des trajectoires utilisées. La figure 6.5b est la post-image de la balise montrée dans la figure 6.5a. La région noire représente la zone qui peut être atteinte par des chemins d'ordre six en partant de cette balise.

L'algorithme $EXPLORE$ consiste à maximiser la distance aux balises précédemment placées en utilisant des trajectoires d'ordre six. La figure 6.6 montre comment l'algorithme $EXPLORE$ place d'une manière successive les balises dans l'espace libre du robot. On remarque la distribution uniforme des balises dans l'espace exploré.

6.1.3 La combinaison des deux fonctions

A chaque étape de l'algorithme $EXPLORE$, on met à jour l'arbre permettant de retrouver facilement le chemin menant de la position à une balise déjà posée. Pour cela, il suffit de noter pour chaque balise nouvellement placée, la balise père et

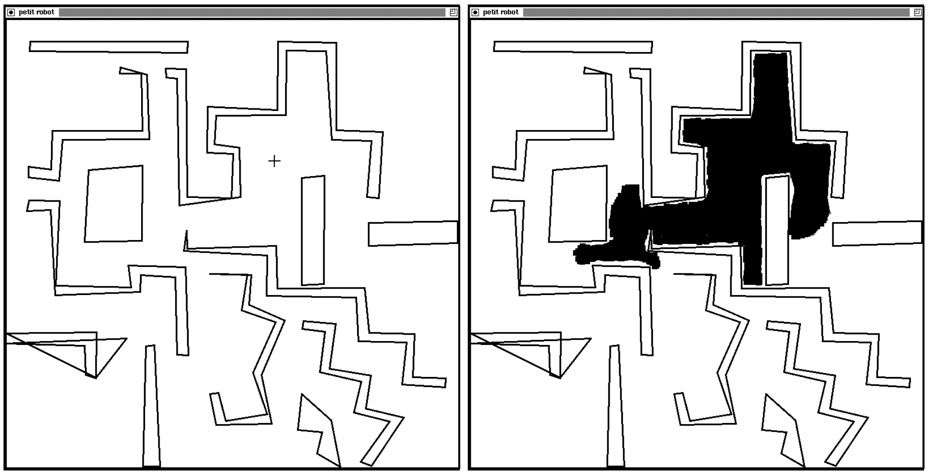


FIG. 6.5 - La post-image produite par une balise.

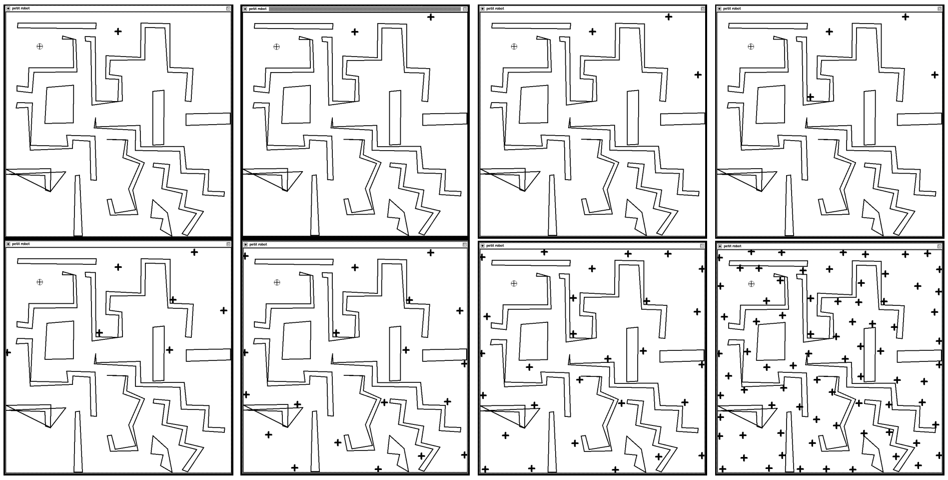


FIG. 6.6 - L'évolution des balises dans l'espace libre.

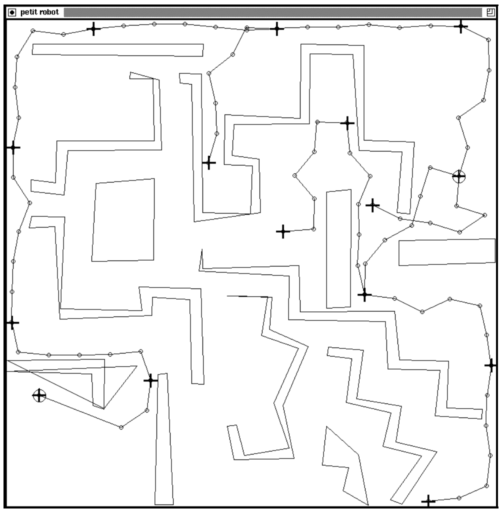


FIG. 6.7 - Le “fil d'Ariane” des balises pour le robot mobile holonome.

la trajectoire permettant de l'atteindre. Un exemple de cet arbre est montré dans la figure 6.7.

Nous re-écrivons ici l'algorithme Fil d'Ariane :

DÉBUT : $L_1 \leftarrow \hat{q}_\circ, EL \leftarrow \{L_1 = \hat{q}_\circ\}, t \leftarrow 1$

1. EXÉCUTION DE SEARCH : Si $SEARCH(L_t, \hat{q}_\bullet) = 0$ alors un chemin a été trouvé.

2. EXÉCUTION D'EXPLORE : Placer une nouvelle balise L_{t+1} avec la fonction $EXPLORE(t)$, $EL \leftarrow EL \cup \{L_{t+1}\}$, $t \leftarrow t+1$ *goto 1.*

Ainsi, soit $\hat{\gamma}_S$ la trajectoire partant de la balise L_k telle que $SEARCH(\hat{q}_\bullet) = 0$. Nous pouvons construire la trajectoire finale $\hat{\gamma}_f$ allant de la configuration initiale $\hat{q}_\circ$ à la configuration finale $\hat{q}_\bullet$ grâce à la procédure suivante :

```
Construire_Solution(k, γ̂_S)
begin
 père = arbre[k].père;
 γ̂_f = γ̂_S;
 while(père ≠ nul)
   γ̂_f = arbre[père].γ̂ * γ̂_f;
   père = arbre[père].père;
 return(γ̂_f);
end
```

Le symbole "$*$" représente l'opération de concaténation de deux chemins. Cette opération a été définie à la page 58.

La figure 6.8 montre deux exemples de trajectoires planifiées par l'algorithme Fil d'Ariane.

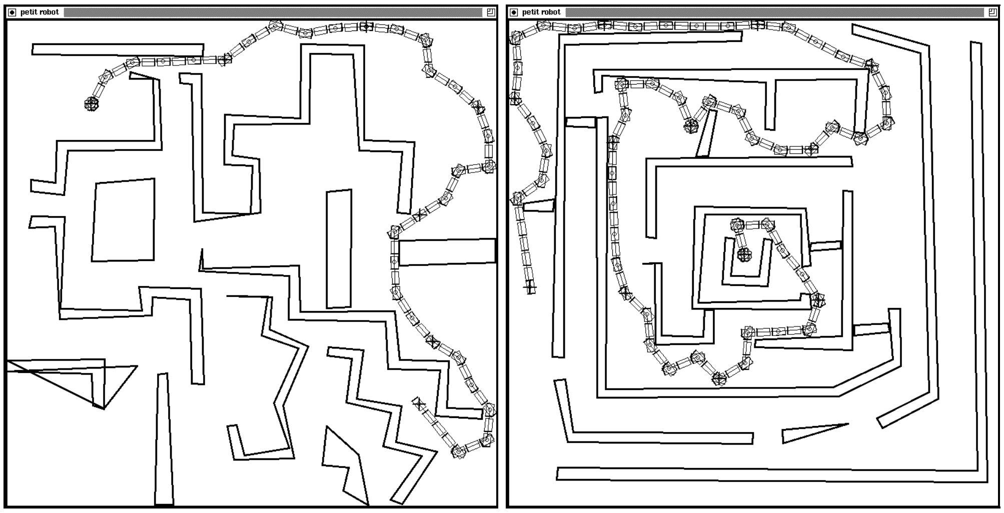


FIG. 6.8 - Deux trajectoires planifiées par l'algorithme Fil d'Ariane.

6.2 Discussion sur le comportement de l'algorithme $EXPLORE$

6.2.1 La décroissance de la fonction $EXPLORE(t)$

Nous avons vu que le domaine de la fonction $EXPLORE$ est l'ensemble de trajectoires d'ordre k partant d'une des balises déjà posées. Il est clair que lorsqu'on augmente l'ordre des trajectoires, on peut augmenter aussi la taille de la post-image engendrée.

L'utilisation d'ordres differérents pour les trajectoires nous amène alors à des comportements différents de la fonction $EXPLORE(t)$. Par exemple, si on note par $EXPLORE_k(t)$ l'application de l'algorithme $EXPLORE$ en utilisant des trajectoires d'ordre k, alors nous nous attendons à ce que pour le même environnement et le même point de départ $EXPLORE_{\ell_1}(t) \leq EXPLORE_{\ell_2}(t)$ si $\ell_1 < \ell_2$.

Un autre facteur qui peut faire varier le comportement de $EXPLORE(t)$ est la technique d'optimisation employée. Il se peut qu'une technique d'optimisation soit meilleure qu'une autre, et par conséquent, qu'elle nous amène à une exploration plus efficace. Dans la suite, nous montrerons sur un ensemble d'expérimentations du robot mobile holonome comment le comportement de $EXPLORE(t)$ varie en

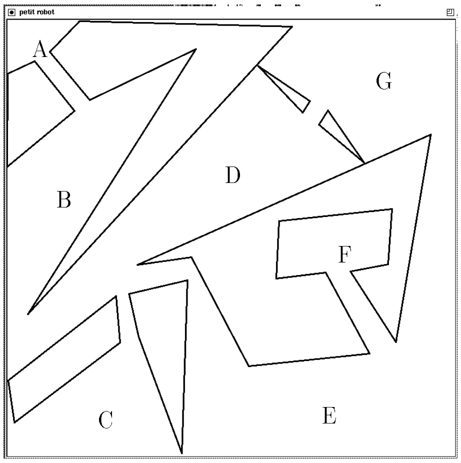


FIG. 6.9 - L'espace à Explorer.

fonction de la longueur des chemins et de la technique d'optimisation utilisée. La figure 6.9 montre l'espace avec lequel nous avons réalisé l'ensemble des expérimentations présentées dans ce paragraphe. Le point de départ ainsi que le nombre de balises placées est constant pour chacun des exemples.

Influence de l'ordre des chemins

L'exemple 1, figure 6.10a, montre les résultats obtenus par $EXPLORE(t)$ en utilisant des trajectoires d'ordre 10. Rappelons que la taille du côté du plus petit cube contenant un espace X est notée par $s(X)$ (voir page 47) la valeur $s(\mathcal{C}_{libre})$ étant connue, la courbe théorique de l'expression 3.5 peut être établie et est également présentée (voir page 54). Nous la réécrivons ci-dessous :

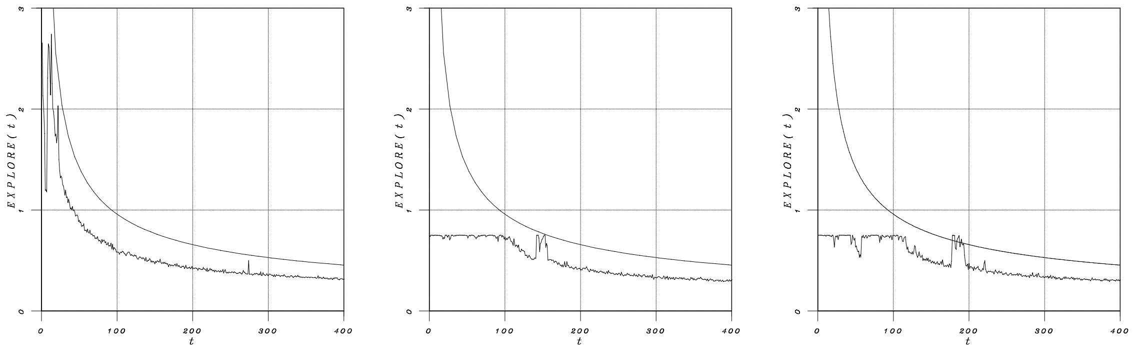
$$\varepsilon_t \leq \frac{2s * \left(\frac{\sigma n'}{t+1}\right)^{\frac{1}{n}}}{1 - 2 * \left(\frac{\sigma n'}{t+1}\right)^{\frac{1}{n}}}$$

L'exemple 2, figure 6.10b, correspond aux résultats obtenus par $EXPLORE(t)$ en utilisant des trajectoires d'ordre 2. Dans les deux exemples, un algorithme génétique a été utilisé comme technique d'optimisation. Après 10 itérations, le meilleur individu d'une population de taille 36 est pris comme solution optimale.

Nous pouvons apprécier que l'utilisation des chemins d'ordre plus grand permet le placement de balises dans des positions plus éloignées. Par conséquent, $EXPLORE(t)$ décroît plus rapidement en utilisant un ordre égal à 10. Par ailleurs, le coût d'optimisation de $EXPLORE$ augmente en proportion avec l'ordre des chemins, un compromis doit être trouvé expérimentalement.

Influence de la technique d'optimisation

L'exemple 3, figure 6.10c, montre le comportement de $EXPLORE(t)$ en utilisant des chemins de même longueur que dans l'exemple 2. La différence réside dans la technique d'optimisation employée. Dans ce cas, on utilise un simple tirage aléatoire : 64 chemins sont engendrés au hasard, le meilleur d'entre eux est pris comme solution optimale. On peut remarquer que le comportement de l'exemple 3 est plus "chaotique" que celui de l'exemple 2. Ce comportement "chaotique" nous amène à une exploration de l'espace libre moins efficace, il nous faut plus de balises pour découvrir des nouvelles régions de l'espace libre. Par exemple, dans les figures 6.10b et 6.10c nous pouvons noter l'apparition d'une irrégularité dans la décroissance d'$EXPLORE(t)$. Cette irrégularité marque la découverte de nouvelles portions de l'espace des configurations non encore explorées ou "chambres" (dans ce cas, le "chambres" E et G). On peut remarquer que la découverte de ces "chambres" est plus tardive dans l'exemple 3. Ceci tend à démontrer l'efficacité des algorithmes génétiques pour ce problème. Dans le paragraphe suivant, nous présentons un exemple dans lequel il est possible d'identifier le passage dans des régions inexplorées de l'espace libre. Ce passage est identifié grâce à la variation de $EXPLORE(t)$.

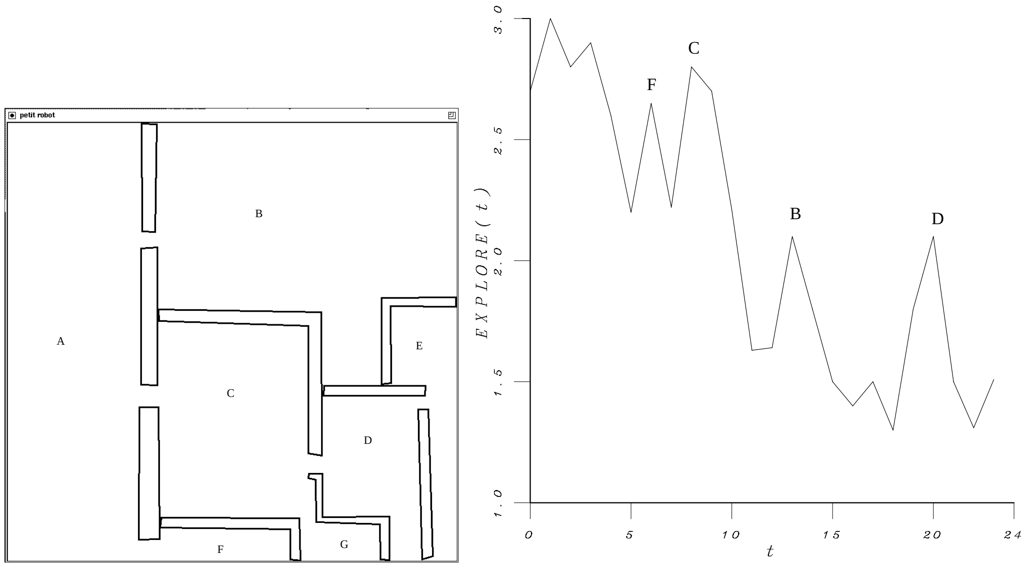


a b c

FIG. 6.10 - Graphiques de $EXPLORE(t)$.

6.2.2 Informations obtenues par EXPLORE(t)

A mesure que le temps passe, $EXPLORE(t)$ nous donne des informations concernant l'exploration de l'espace libre. Au contraire de $EXPLORE_{\infty}(t)$ (la courbe théorique), la fonction $EXPLORE(t)$ peut rester constante, diminuer ou augmenter avec le temps. Lorsque $EXPLORE(t)$ reste constant ou diminue nous pouvons conclure qu'au pire des cas, l'algorithme EXPLORE se trouve dans un processus de saturation. Saturer une "chambre" revient à remplir celle-ci avec suffisamment de balises pour trouver une nouvelle issue. Pour ce faire, l'algorithme doit diminuer la résolution de l'exploration pour découvrir un passage plus petit que la résolution actuelle. A un moment donné, $EXPLORE(t)$ arrive à une résolution suffisamment petite pour traverser ce passage; la valeur de $EXPLORE(t)$ augmente alors de manière considérable. A ce moment, une nouvelle "chambre" de l'espace libre est découverte et il faut commencer un nouveau processus de saturation.



a b

FIG. 6.11 - L'environnement exploré et le graphique $EXPLORE(t)$ obtenu.

Les figures 6.11a et 6.11b montrent respectivement un environnement et les résultats obtenus par $EXPLORE(t)$. On peut remarquer que la première balise dans les pièces B, C, et D a provoqué des changement significatifs dans la valeur de $EXPLORE(t)$. En observant ces valeurs, nous pouvons envisager une amélioration de la fonction d'évaluation. Si la valeur de $EXPLORE(t)$ augmente alors il n'est plus nécessaire de considérer les balises précédemment posées avec une valeur inférieure. Cette amélioration a pour but de diminuer la dimension de l'espace de recherche.

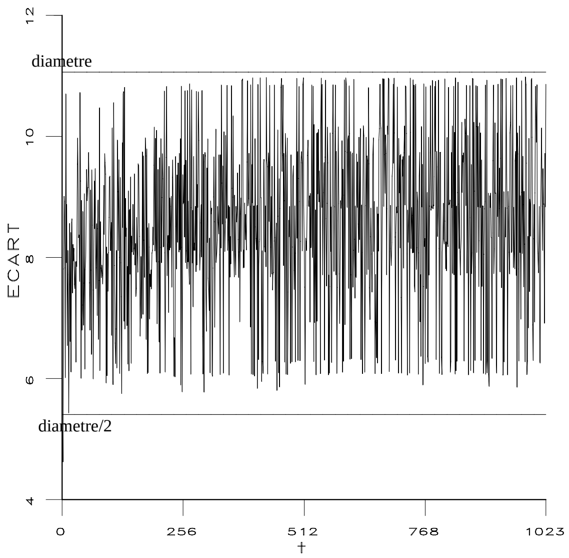


FIG. 6.12 - L'écart de L_t.

6.2.3 Où ont été placées les balises ?

Lorsque l'algorithme $EXPLORE$ sélectionne une nouvelle balise, le placement de celle-ci dépend de la position des balises précédemment posées. Une exploration uniforme de l'espace libre implique un placement de balises dans la frontière de $\mathcal{C}_{libre}$ et un placement à l'intérieur. Ainsi, soit EL_t l'ensemble de balises au temps t on définit l'écart de L_{t+1} comme :

$$\max_{L_i \in EL} d(L_{t+1}, L_i)$$

Par définition l'écart de la balise L_{t+1} ne peut pas être plus grand que la valeur du diamètre $\delta(\mathcal{C}_{libre})$ de l'espace libre. D'autre part, l'écart ne peut pas être plus petit que $\delta(\mathcal{C}_{libre})/2$.

La figure 6.12 montre les écarts de chacune des balises obtenues par $EXPLORE(t)$ Dans cette figure on voit que l'écart des balises varie effectivement dans l'intervalle $[\delta(\mathcal{C}_{libre})/2, \delta(\mathcal{C}_{libre})]$. L'écart nous montre que l'algorithme $EXPLORE$ place des balises dans des régions différentes de l'espace libre. Cette caractéristique a des conséquences importantes pour l'algorithme $SEARCH$. Ce placement uniformément distribué des balises permet à l'algorithme SEARCH de sortir systématiquement des minima locaux.

Chapitre 7

Planification de trajectoires pour un robot à six degrés de liberté

Nous présentons dans ce chapitre une application de l'algorithme Fil d'Ariane implantée sur une machine massivement parallèle : un système de planification de trajectoires "en ligne" pour un bras manipulateur à six degrés de liberté évoluant dans un environnement dynamique.

L'objectif est de planifier des trajectoires pour un bras (*robot I*) avec l'algorithme Fil d'Ariane alors qu'un autre bras évolue indépendamment dans le même espace de travail. Le planificateur doit être suffisamment rapide pour re-planifier une nouvelle trajectoire vers le but, lorsque le deuxième robot (*robot II*) change de configuration et empêche le *robot I* d'exécuter la trajectoire précédemment calculée.

Dans la première partie de ce chapitre, nous présentons une description générale du système de planification de trajectoires. Dans la deuxième partie, nous introduisons la méthode de modélisation du robot et de l'environnement de travail. Puis, nous décrivons le calcul de débattements basé sur ce modèle. Finalement, le codage et décodage des trajectoires utilisés par $SEARCH$ et $EXPLORE$ sont présentés. L'implantation parallèle de l'algorithme Fil d'Ariane fait l'objet du chapitre suivant.

7.1 Description générale du système

Nous avons expérimenté l'algorithme Fil d'Ariane en planifiant des trajectoires pour un bras SCEMI à six degrés de liberté.

La figure 7.1 représente l'architecture de notre site expérimental. Elle est composée de trois grands sous-ensembles :

1. deux robots équipés chacun d'un système de contrôle-commande **KALI**[1],
2. une machine parallèle **Méga-Node** constituée de 128 transputers de type **T800**,
3. un système de modélisation géométrique **ACT** .

Le *robot I* est sous contrôle du Méga-Node qui exécute une version parallèle de l'algorithme Fil d'Ariane. Le *robot II* est utilisé comme un obstacle dynamique : il est contrôlé par un générateur de trajectoires aléatoires; il peut bloquer le *robot I* alors que celui-ci exécute une trajectoire. Les bras sont pilotés indépendamment par deux armoires de commande utilisant chacune le logiciel KALI. Pour communiquer les trajectoires calculées et lire la position des robots, nous utilisons un utilitaire de communication spécialement conçu pour communiquer avec ACT: RobotCom[2]. Dans ce montage expérimental, l'armoire de commande du robot sous le contrôle du planificateur ne communique pas directement avec le Méga-Node. Toutes les trajectoires transitent d'abord par le serveur Sun 4 puis par une station de travail Silicon Graphics supportant le logiciel ACT.

L'implantation de l'algorithme Fil d'Ariane repose sur trois niveaux de parallélisme. Le premier niveau est l'exécution parallèle des algorithmes SEARCH et EXPLORE. D'une part, EXPLORE produit des balises, d'autre part SEARCH les exploite pour atteindre la configuration finale. Un deuxième niveau de parallélisme a été implanté pour les algorithmes SEARCH et EXPLORE. Ces algorithmes sont tous les deux des algorithmes d'optimisation implantés sous la forme d'algorithmes génétiques parallèles. Enfin, le troisième niveau de parallélisation se trouve dans la fonction d'évaluation utilisée par les deux algorithmes génétiques.

L'implantation parallèle de ces trois niveaux est décrite en détail dans le chapitre 8.

[1] Kali est un système de contrôle et de commande de robots, initialement développé par l'université de McGill au Canada et actuellement commercialisé par Aleph Technologies.

[2] ACT et RobotCom sont des produits développés et commercialisés par Aleph Technologies

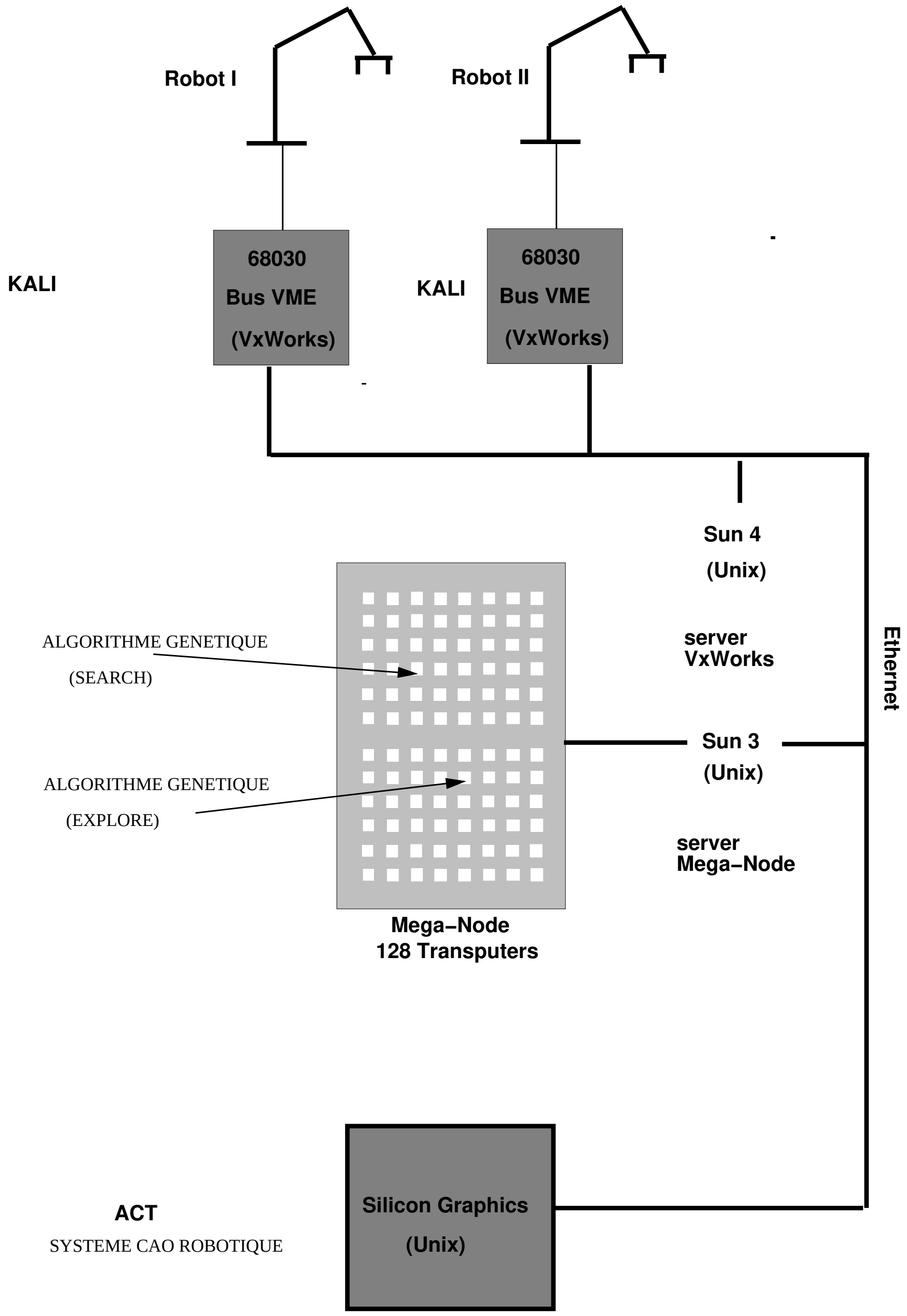


FIG. 7.1 - Architecture du système.

7.1.1 Utilisation du système

L'utilisation du système se déroule comme suit :

Premièrement, le système ACT est utilisé pour décrire la scène avec les deux bras et les obstacles statiques en définissant une cellule robotique. Ensuite, on compile automatiquement cette cellule pour la transformer dans une représentation simplifiée appelée le *modèle optimisé*. Les configurations initiale et finale sont alors données pour le *robot I*, le Méga-Node calcule une trajectoire pour ce robot. Le temps de calcul d'une trajectoire sans collision est d'environ 2 secondes. Cette trajectoire est envoyée à la station de travail Silicons Graphics où elle est simplifiée avant d'être envoyée au *robot I*. Tandis que la trajectoire s'exécute, le générateur de trajectoires aléatoires du *robot II* peut engendrer un mouvement avec une probabilité définie par l'utilisateur. Dès lors qu'un mouvement est exécuté par le *robot II*, le *robot I* interrompt sa trajectoire. Les nouvelles configurations des robots sont alors envoyées au Méga-Node qui calcule une autre trajectoire. Cette boucle continue jusqu'à ce que le *robot I* arrive à la position finale; à ce moment là, une nouvelle configuration but peut être définie.

7.2 Le système de modélisation

Le système de représentation de l'univers du robot est construit en trois étapes :

1. Utilisation du système ACT pour la description de la scène réelle, avec le ou les robots et les obstacles, voir figures 7.2 et 7.3.

2. Transformation de cette représentation en un modèle optimisé. La scène est représentée comme un ensemble de parallélépipèdes (voir figure 7.4). Ce modèle est utilisé pour réaliser tous les calculs mathématiques nécessaires à la planification et à la génération de trajectoires des robots.

3. Finalement, un troisième niveau de représentation est obtenu à partir du deuxième en calculant une sphère englobante pour chacun des parallélépipèdes. L'objectif de ce dernier niveau est de pouvoir tester très rapidement les trajectoires engendrées. Ainsi, lors de l'évaluation d'une trajectoire, il est possible de filtrer de façon quasi-instantanée l'absence de collisions entre le robot et son environnement.

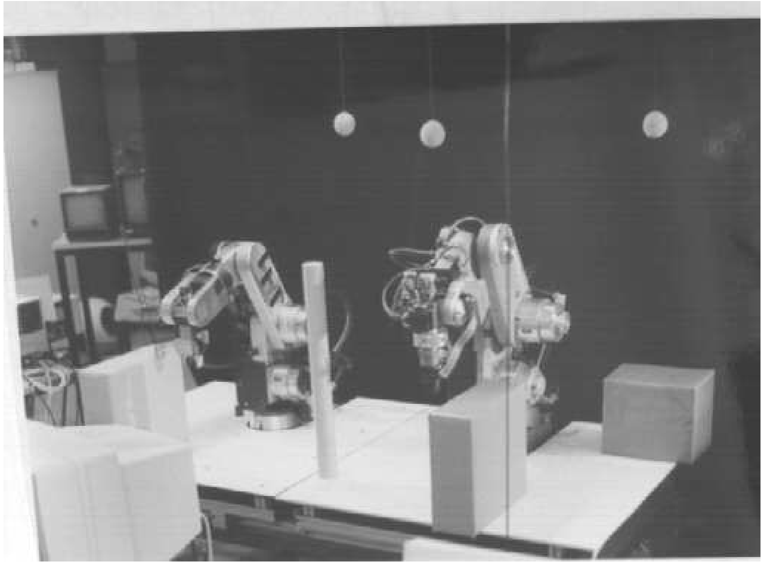


FIG. 7.2 - Scène réelle du site expérimental du LIFIA.

7.2.1 Le premier niveau de modélisation et le système ACT

L'origine de ACT ne se trouve pas dans l'extension d'un système de CAO classique mais résulte de recherches effectuées dans le domaine de la programmation automatique des robots. Dans cette optique, ACT est d'abord conçu comme une structure d'accueil pour planificateurs, il possède bien entendu les fonctionnalités des systèmes de CAO robotique standard. Dans notre application ACT est utilisé :

1. pour décrire les robots, les obstacles et leurs positions relatives,
2. pour obtenir l'information nécessaire qui nous permet de créer un modèle adapté à notre planificateur de trajectoires (modèle optimisé),
3. pour simuler les résultats de la planification.

Modélisation de la scène

La figure 7.2 représente l'une des scènes réelles utilisées pour tester notre planificateur. Cette scène est une vue du site expérimental du LIFIA. Grâce au système ACT, l'utilisateur peut construire un modèle de la scène à partir de primitives géométriques simples. ACT permet aussi de mémoriser toutes les informations cinématiques qui caractérisent la scène. Nous obtenons ainsi le premier niveau de

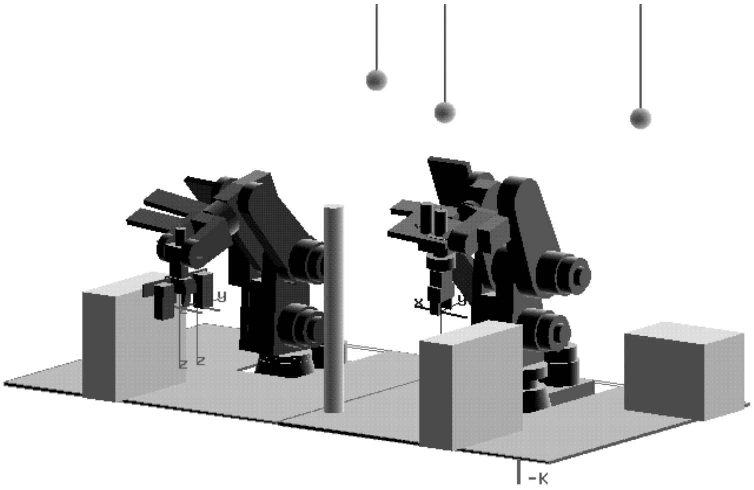


FIG. 7.3 - Premier niveau de modélisation.

modélisation. La figure 7.3 montre l'équivalent de la scène réelle de la figure 7.2. Ce modèle est une réplique "exacte" d'une scène expérimentale réelle[3].

7.2.2 Création du modèle optimisé (deuxième niveau)

Un compilateur d'environnement a été intégré au système ACT standard. Le rôle de ce compilateur est de transformer la représentation interne de la scène utilisée par ACT en une représentation adaptée à notre problème. Dans notre cas, nous avons choisi de représenter les obstacles et les corps du robot par des parallélépipèdes qui les englobent. Le compilateur d'environnement que nous avons implanté transforme donc la scène initiale en une scène où tous les objets sont des parallélépipèdes. La figure 7.4 représente le modèle optimisé de l'environnement de la figure 7.3.

Le modèle optimisé utilise un seul parallélépipède pour englober chacune des composantes [4] du robot. Chaque obstacle est aussi représenté par un parallélépipède [52]. Cependant, il est possible d'utiliser un niveau plus fin de représentation : autrement dit, il est possible d'utiliser plusieurs parallélépipèdes pour représenter chacun des composants du robot et des obstacles.

[3] Dans la figure 7.3 on remarque quelques éléments "inexistants" dans la scène réelle. En effet, ces éléments sont une représentation "vague" des câbles du robot.

[4] Nous entendons par "composante du robot" l'ensemble des corps qui appartiennent à la même articulation.

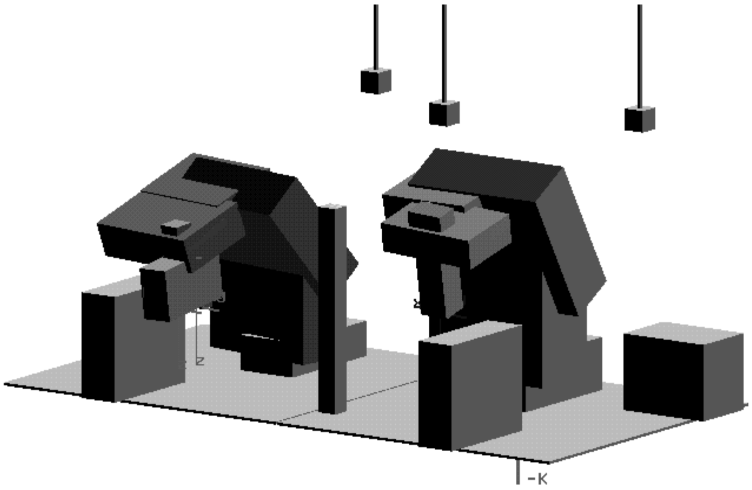


FIG. 7.4 - Deuxième niveau de modélisation.

L'information contenue dans le modèle optimisé est la suivante :

- le nombre d'articulations,
- les dimensions de chacun des composantes du robot (x, y, z),
- la matrice de transformation du corps de la composante i au repère R_i (notée Tc_i),
- la matrice de transformation du repère Ri au repère R_{i-1} (noté $Tm_{i-1,i}$) ou bien au repère du monde pour le repère R_0,
- les butées mécaniques de chaque degré de liberté,
- la configuration initiale,
- le nombre d'obstacles,
- les dimensions des obstacles (x, y, z),
- la matrice de placement de chacun des obstacles (notée To_i).

La figure 7.5 montre l'utilisation des différentes matrices de transformation.

Dans la suite de ce paragraphe, nous décrivons la représentation des parallélépipèdes décomposés en entités géométriques.

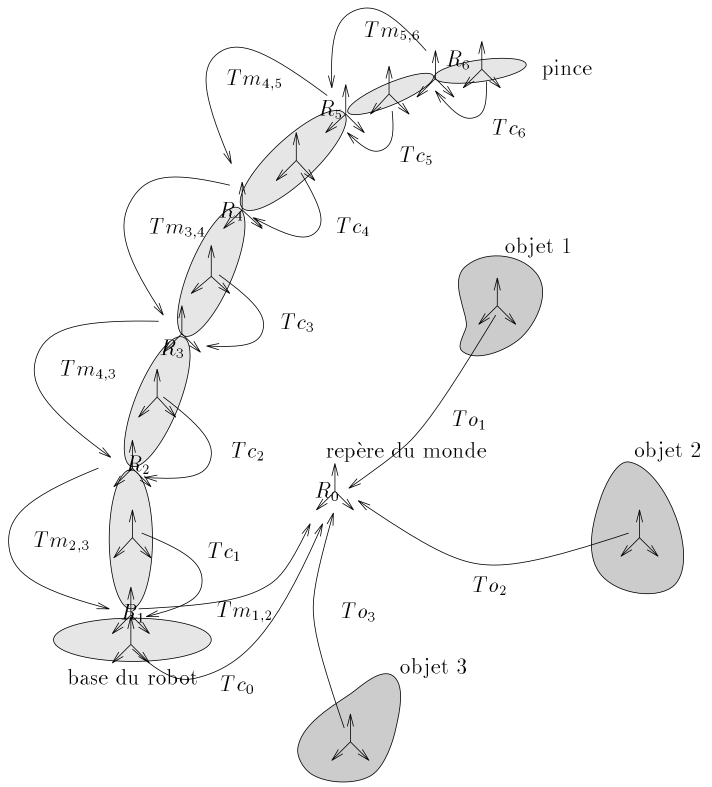


FIG. 7.5 - La chaîne articulaire du robot et les matrices de transformation.

Représentation des parallélépipèdes

Comme nous l'avons déjà dit, le modèle optimisé est composé principalement d'un ensemble de parallélépipèdes, chacun décrit par un ensemble de sommets, d'arêtes et de faces. Ainsi un robot à n degrés de liberté (ou un obstacle) est décomposé géométriquement de la manière suivante :

$$Robot = \{x_1, x_2, \ldots . x_{n+1}\}$$

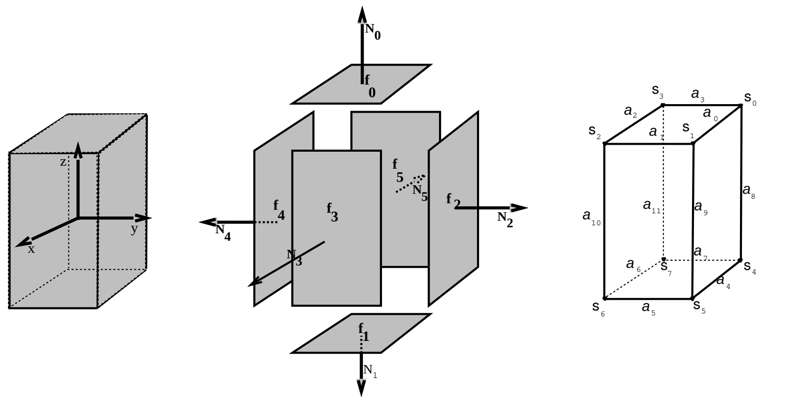


FIG. 7.6 - Décomposition d'un parallélépipède.

où x_i représente un parallélépipède du modèle optimisé. Chacun de ces parallélépipède est alors sous la forme :

$$x_i = \{S_i, A_i, F_i\}$$

où S_i est l'ensemble des sommets, A_i l'ensemble des arêtes et F_i l'ensemble des faces (voir figure 7.6). Ces ensembles sont finalement décomposés de la manière suivante :

$$S_i = \{s_{i,0}, s_{i,1}, \ldots, s_{i,7}\}, \quad A_i = \{a_{i,0}, a_{i,1}, \ldots, a_{i,11}\}, \quad F_i = \{f_{i,0}, f_{i,1}, \ldots, f_{i,5}\} \tag{7.16}$$

La représentation des sommets, arêtes et faces est obtenue à partir de trois types d'entités géométriques différentes : des points, des droites et des plans. Ainsi nous représentons :

1. **Les sommets** : par un point.
2. **Les arêtes** : par deux points et une droite.
3. **Les faces** : par quatre points, quatre droites et un plan.

Ces entités géométriques sont représentées dans l'espace par leurs coordonnées de Plücker[54] (voir annexe B).

7.2.3 Le modèle sphérique (troisième niveau)

Le dernier niveau de modélisation du système consiste à représenter chacun des parallélépipèdes du modèle optimisé par une sphère qui les englobe [62]. L'avantage de ce niveau de représentation est que les sphères sont représentées par leur rayon et par leur centre. De cette manière, le modèle est exprimé sans difficulté majeure dans tout repère donné. Lors du test de collision, ce modèle permet de filtrer certains objets ou composantes du robot par un test simple. Ainsi, le nombre d'objets intervenant dans le calcul de débattement diminue car les objets, en dehors de la trajectoire des objets en mouvement sont éliminés rapidement par ces filtres.

En résumé, pour réaliser le calcul des débattements, nous devons d'abord créer un modèle CAO du ou des robots et de l'environnement, puis "compiler" ce modèle dans un modèle qui nous est propre. Dès lors, il faut distinguer le modèle interne du planificateur (boîte englobante) et la représentation graphique de ACT.

7.3 Le calcul de débattements

Dans la procédure de décodage d'une trajectoire du robot, présentée dans le dernier paragraphe de ce chapitre, la valeur Δ_i^j signifie un déplacement de l'articulation i et par conséquent un test de collision entre un ensemble X d'objets mobiles (le robot) et un ensemble Y d'objets statiques (les obstacles). Etant donné l'axe de rotation de l'articulation i en mouvement, il faut calculer la valeur du mouvement circulaire maximal décrit par chaque objet en X avant d'entrer en collision avec un objet en Y : cette valeur est appelée le débattement. Il est clair que le calcul de débattements est nécessaire pour évaluer une trajectoire.

Ainsi, étant donné le repère de rotation R_i de l'articulation en mouvement, il nous faut tout d'abord exprimer tout les objets des ensembles X et Y dans le repère[5] R_i. Ensuite, on effectue le calcul de débattement pour chaque couple (x, y) du produit cartésien $X \times Y$. La valeur la plus petite de ces débattements est le débattement maximal de l'ensemble X par rapport à Y.

La fonction de calcul de débattement *positif* entre le parallélépipède en mouvement x et le parallélépipède statique y est notée $\psi(x, y)$. Cette fonction représente la valeur maximale du mouvement de rotation *positive* de x par rapport à y. D'une manière similaire, nous définissons la fonction $\Gamma(X, Y)$ représentant la valeur maximale du mouvement de rotation *positive* pour un ensemble d'objets en mouvement X avec un autre ensemble Y d'objets statiques ainsi :

[5] Nous utilisons comme axe de rotation l'axe z du repère.

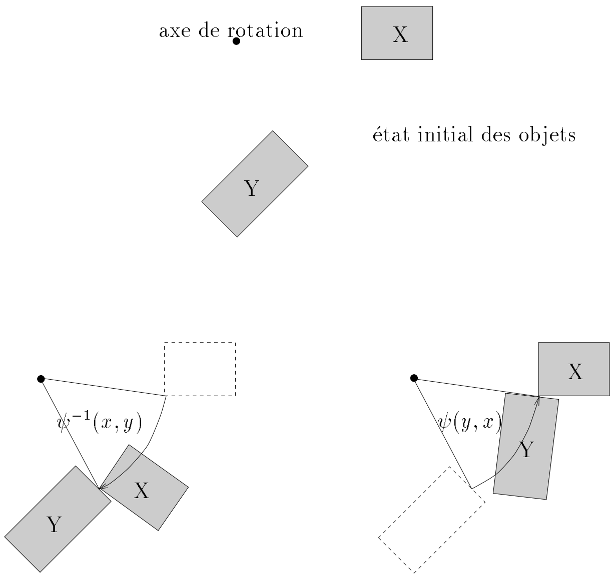


FIG. 7.7 - Equivalence des *valeurs* de $\psi^{-1}(x, y)$ et $\psi(y, x)$.

$$\Gamma(X, Y) = \min_{x \in X, y \in Y} \psi(x, y) \tag{7.17}$$

On peut remarquer que la *valeur* de rotation maximale négative notée $\psi^{-1}(x, y)$ est égale à $\psi(y, x)$ (voir figure 7.7). Par conséquent :

$$\Gamma^{-1}(X, Y) = \Gamma(Y, X)$$

7.3.1 Calcul de débattements entre deux parallélépipèdes

Un parallélépipède x effectuant un mouvement de rotation peut entrer en collision avec un autre parallélépipède y par trois types différents de contact [49][51] :

1. **Type A (point-face) :** Un sommet de l'objet mobile x contre une face de l'objet fixe y.
2. **Type B (face-point) :** Un sommet de l'objet fixe y contre une face de l'objet mobile x .
3. **Type C (arête-arête) :** Une arête de l'objet mobile x contre une arête de l'objet fixe y .

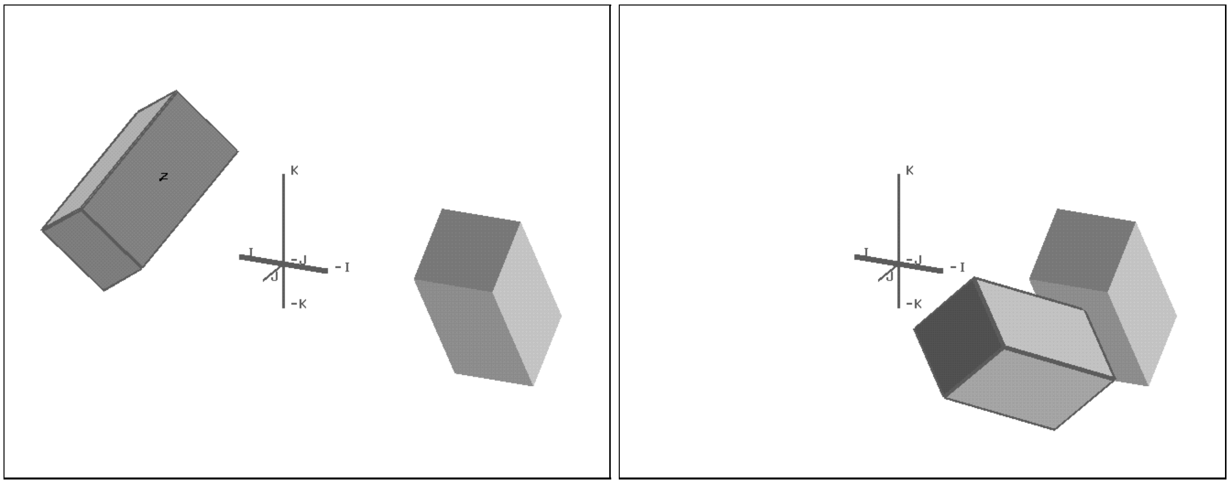


FIG. 7.8 - Contact type A.

Les figures 7.8, 7.9 et 7.10 montrent les trois types de contacts. En résumé pour deux parallélépipèdes x (objet mobile) et y (objet fixe) :

- les contacts de type A sont détectés pour chaque sommet $s_i \in S \in x$ par un calcul d'interférence entre le *lieu géométrique* décrit par le sommet s_i dans son mouvement de rotation et les faces $\{f_0, f_1, \ldots, f_5\}$ du parallélépipède y.
- de même, les contacts du type B sont détectés pour chaque sommet $s_j \in S \in y$ par le calcul d'interférence entre le *lieu géométrique* décrit par le sommet s_j dans son mouvement de rotation et les faces $\{f_0, f_1, \ldots, f_5\}$ du parallélépipède x.
- enfin, les contacts de type C sont détectés pour chaque arête $a_i \in A \in x$ par un calcul d'interférence entre le *lieu géométrique* décrit par le mouvement de a_i et les arêtes $\{a_0, a_1, \ldots, a_{12}\}$ de y.

Par conséquent le calcul du débattement $\psi(x, y)$ requiert :

- $6 * 8 = 48$ calculs de type A,
- $8 * 6 = 48$ calculs de type B,
- $12 * 12 = 144$ calculs de type C.

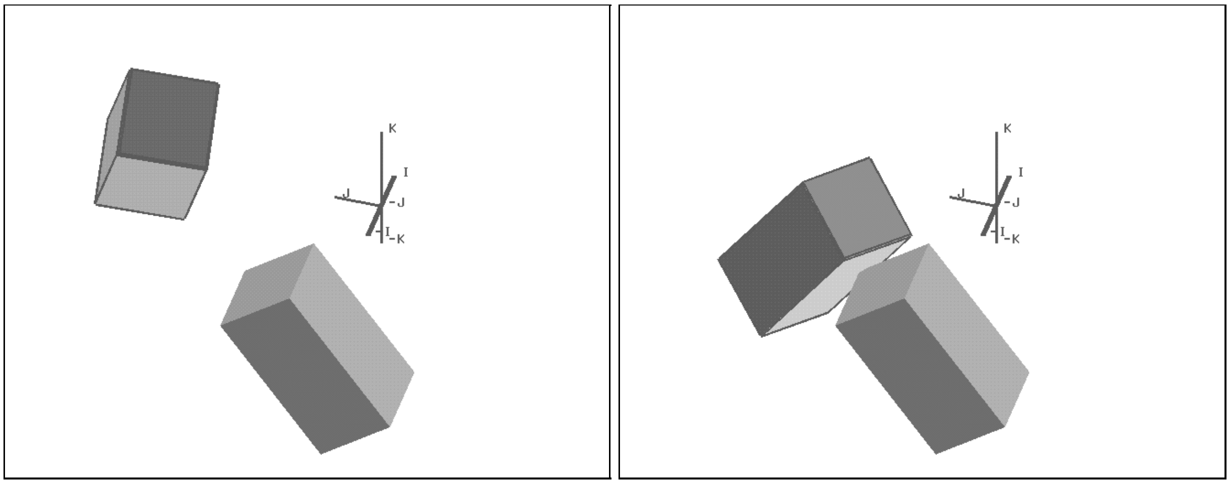


FIG. 7.9 - Contact type B.

soit 240 calculs élémentaires pour la fonction $\psi(x, y)$ et $(card(X \times Y) * 240)$ pour $\Gamma(X, Y)$. Etant donné l'aspect symétrique d'un débattement, il nous suffit de considérer les contacts de type A et les contacts de type C ; les calculs sur les contacts de type B se déduisent de ceux de type A. Ainsi $\psi(x, y)$ est obtenue grâce aux fonctions :

- $\psi_A(sommet, face)$ pour les contacts de type A dans le sens positif de rotation,
- $\psi_B(face, sommet)$ (déduit à partir de ψ_A) pour les contacts de type B dans le sens positif de rotation.
- et ψ_C(arête,arête) pour les contacts de type C dans le sens positif de rotation.

Grâce à ces trois fonctions, nous obtenons trois angles qui représentent respectivement le débattement maximal des points, des faces et des arêtes d'un objet en mouvement x_i par rapport à un autre, statique, y_j :

$$\theta_A = \min_{s_{i,k} \in S_i, f_{j,l} \in F_j} \psi_A(s_{i,k}, f_{j,l})$$
$$\theta_B = \min_{f_{i,k} \in F_i, s_{j,l} \in S_j} \psi_B(f_{i,k}, s_{j,l})$$
$$\theta_C = \min_{a_{i,k} \in A_i, a_{j,l} \in A_j} \psi_C(a_{i,k}, a_{j,l})$$

Ainsi, le débattement $\psi(x_i, y_j)$ est calculé comme :

$$\psi(x_i, y_j) = \min\{\theta_A, \theta_B, \theta_C\}$$

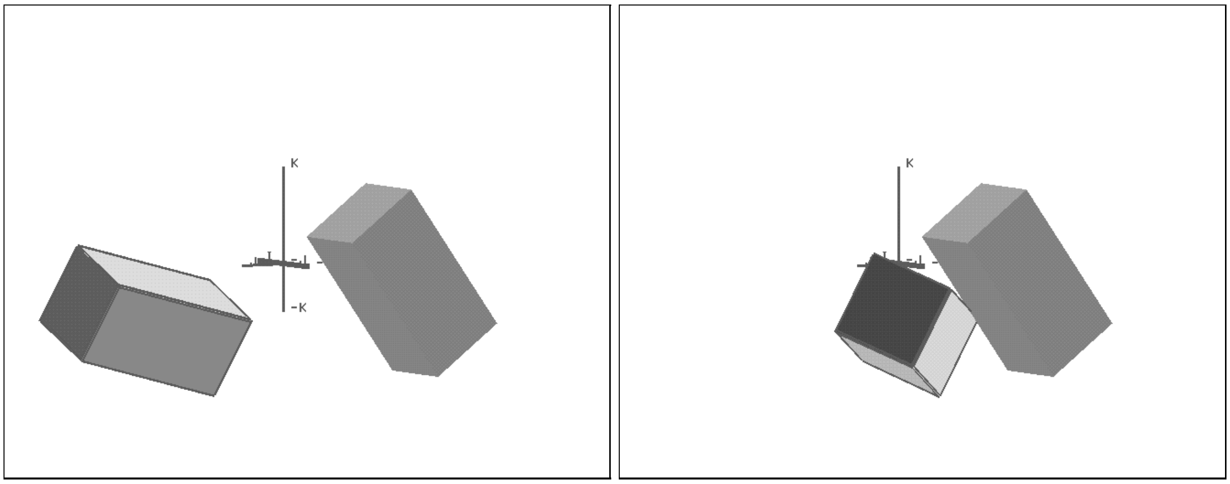


FIG. 7.10 - Contact type C.

7.3.2 Mise à jour de la position du robot et des obstacles

Comme nous l'avons déjà vu, le ou les robots et les obstacles sont représentés par un ensemble de parallélépipèdes :

$$\begin{aligned} Robot &= \{x_{r_1}, x_{r_2}, \ldots, x_{r_{n+1}}\} \\ Obstacles &= \{x_{o_1}, x_{o_2}, \ldots, x_{o_l}\} \end{aligned}$$

où x_{r_i} représente le parallélépipède englobant la composante i, x_{o_i} représente le parallélépipède englobant l'obstacle i, n est le nombre de degrés de liberté et l le nombre d'obstacles. Par conséquent à partir du moment où le degré de liberté i bouge, l'ensemble de parallélépipèdes $X = \{x_{r_i}, x_{r_{i+1}}, x_{r_{i+2}}, \ldots, x_{r_{n+1}}\}$ entre en mouvement. L'ensemble X peut alors entrer en collision avec les objets $\{x_{o_1}, x_{o_2}, \ldots, x_{o_l}\}$ et avec un ensemble $X_r \subset \{x_{r_0}, x_{r_1}, \ldots, x_{r_{i-1}}\} \subset Robot$ si les composants en mouvement du robot peuvent toucher un des composants statiques du robot. Ainsi l'ensemble Y d'objets statiques est égal à :

$$Y = \{x_{o_1}, x_{o_2}, \ldots, x_{o_l}\} \cup X_r$$

et l'ensemble X d'objets en mouvement à :

$$X = \{x_{r_i}, x_{r_{i+1}}, x_{r_{i+2}}, \ldots, x_{r_{n+1}}\}$$

Dès lors qu'un débattement $\psi(x, y)$ est calculé, il faut tout d'abord exprimer les objets x et y dans le repère R_i de l'articulation en mouvement. Cependant, dans le modèle optimisé chacune des entités géométriques des obstacles est exprimée dans le repère du monde. D'autres, celles appartenant aux composantes du robot, sont exprimées par rapport au repère de la composante d'appartenance.

Exprimer tous les objets des ensembles X et Y dans le même repère implique le calcul des matrices de transformation homogène sous la forme $M_{i,j}$ où i est l'indice du repère de rotation et j celui du repère de l'objet. Cette matrice est sous la forme suivante [67] :

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.18}$$

Par exemple, pour exprimer le parallélépipède représentant la pince du robot dans un repère R_i, il nous faut une matrice de transformation $M_{i,6}$, c'est-à-dire une transformation de R_6 à R_i. Ainsi, lorsqu'on bouge l'articulation i, nous avons besoin de trois groupes de matrices :

- les premières matrices $\{M_{i,i+1}, M_{i,i+2}, \ldots, M_{i,n}\}$ transformant tous les objets en mouvement $\{x_{r_{i+1}}, x_{r_{i+2}}, \ldots, x_{r_n}\}$ dans le repère R_i,
- les deuxièmes $\{M_{i,0}, M_{i,1}, \ldots, M_{i,i-1}\}$ transformant dans le repère R_i l'ensemble X_r des composantes statiques du robot en risque de collision avec les composantes en mouvement du robot .
- finalement, la matrice $M_{i,0}$ utilisée pour transformer tous les obstacles dans le repère R_i.

Chacune de ces matrices est obtenue par une suite de multiplications de matrices $M_{i-1,i}$ et les inverses de celles-ci. Pour obtenir ces matrices d'une manière efficace, nous avons développé une méthode originale permettant de maintenir à jour les positions relatives le long d'un chemin Manhattan; cette méthode est décrite ci-après.

Transformations homogènes des composantes du robot dans les repères des axes de rotation

Pour maintenir à jour cet état du monde, nous avons utilisé une technique tenant compte des propriétés de la chaîne articulaire qui représente le robot. Une matrice contenant des transformations homogènes maintient l'état du robot. Nous appelons cette matrice la matrice "$\mathcal{M}$". Les caractéristiques de cette matrice pour un robot à n degrés de liberté sont les suivantes :

- $\mathcal{M}$ est de dimension $(n+1) \times (n+1)$.
- Un élément $M_{i,j}$ de $\mathcal{M}$ contient la matrice de transformation du repère R_j au repère R_i.
- $M_{i,j} = M_{j,i}^{-1}$.
- si $i = j$ alors $M_{i,j} = I$.
- Tout élément $M_{i,j}$ de $\mathcal{M}$ peut être exprimé comme une suite de multiplications de matrices sous la forme $M_{i,j} = M_{i,i+1} * M_{i+1,i+2} * \ldots M_{j-1,j}$ pour $i < j$ et sous la forme $M_{i,j} = M_{i,i-1} * M_{i-1,i-2} * \ldots M_{j+1,j}$ pour $i > j$. Par exemple $M_{1,4} = M_{1,2} * M_{2,3} * M_{3,4}$.

En général, étant donné une matrice sous la forme de l'expression 7.18, l'inverse est [67] :

$$\begin{bmatrix} n_x & n_y & n_z & -\hat{p}\cdot\hat{n} \\ o_x & o_y & o_y & -\hat{p}\cdot\hat{o} \\ a_x & a_y & a_z & -\hat{p}\cdot\hat{a} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.19}$$

où "$\cdot$" représente le produit scalaire et $\hat{p}$,$\hat{n}$,$\hat{a}$ et $\hat{o}$ les quatre vecteurs formés par les colonnes de l'expression 7.18, c'est-à-dire $\hat{p} = (p_x, p_y, p_z, 1)$, $\hat{n} = (n_x, n_y, n_z, 0)$, etc. Ainsi, la matrice $M_{j,i}$ peut être calculée à partir de $M_{i,j}$. Etant donné ces dernières caractéristiques, il est facile de calculer la matrice $\mathcal{M}$ à partir des éléments élémentaires $M_{i,i+1}$ pour $i = 0, 1, \ldots, n$.

Etant donné une configuration du robot $\hat{q} = (\theta_0, \theta_1, \ldots, \theta_{n-1})$, nous pouvons calculer $\mathcal{M}$ avec l'algorithme suivant :

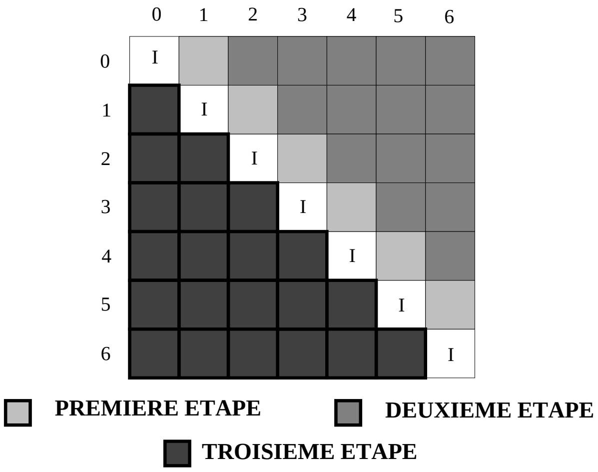


FIG. 7.11 - Etapes du calcul de la matrice $\mathcal{M}$.

CaculMatrice($\mathcal{M}$)
begin
/*Etape 1 : calcul des matrices élémentaires*/
for i from 0 to n-1
    $M_{i,i+1} = Tm_{i,i+1} * R_z(\theta_i)$
/*Etape 2 : calcul des matrices composées*/
  for i from 0 to n-1
    for j from i+2 to n
      $M_{i,j} = M_{i,j-1} * M_{j-1,j}$
/*Etape 3 : calcul des matrices inverses*/
  for i from 0 to n
    for j from i+1 to n
      $M_{j,i} = M_{i,j}^{-1}$
end

où $Tm_{i,i+1}$ est la transformation du repère R_{i+1} du robot au repère R_i et $R_z(\theta_i)$ la matrice de rotation en z de θ_i. La figure 7.5 montre ces repères et la figure 7.11 les différentes étapes de remplissage de $\mathcal{M}$.

Lorsqu'une seule valeur $\theta_{k=0,1,\ldots n-1}$ change, il est très facile de recalculer la matrice $\mathcal{M}$. L'algorithme nécessaire pour exécuter cette mise à jour est le suivant :

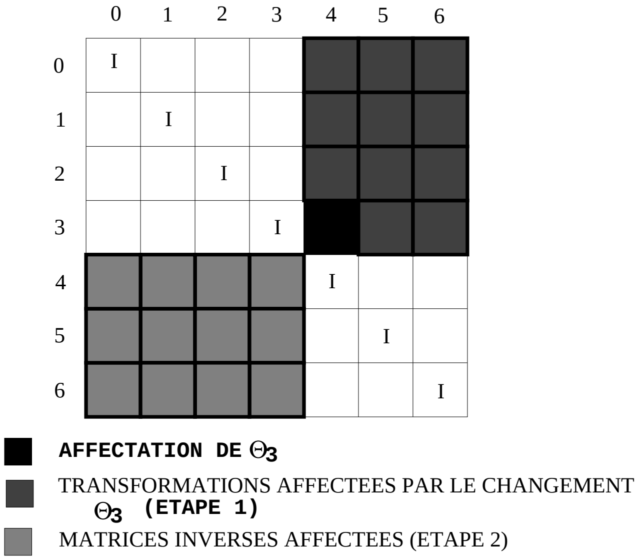


FIG. 7.12 - Mise à jour de la matrice $\mathcal{M}$.

ModifierMmatrice($\mathcal{M}$,k,θ_k)
begin
    for i from k-1 downto 0
        for j from k to n
            if (i = k-1 and j = k)
        /* Etape 1 : recalcul des matrices affectées */
                $M_{i,j} = Tm_{k-1,k} * R_z(\theta_k)$
            else
                $M_{i,j} = M_{i,j-1} * M_{j-1,j}$
        /*Etape 2 : Calcul des matrices inverses affectées*/
            $M_{j,i} = M_{i,j}^{-1}$
end

Les étapes de la mise à jour avec cet algorithme sont montrées dans la figure 7.12.

A l'aide de la matrice $\mathcal{M}$ nous pouvons alors exprimer les objets des ensembles X et Y dans le repère de rotation R_i et effectuer le calcul de débattements correspondants. Lorsque le robot a effectué le mouvement en θ_i, il est possible de mettre à jour la matrice $\mathcal{M}$ et de calculer le prochain débattement.

7.3.3 Transformations homogènes des obstacles dans les repères des axes de rotation

Une fois que la matrice $\mathcal{M}$ a été calculée, il suffit d'exprimer l'ensemble Y dans le repère du monde et de le transformer avec les éléments $M_{i,0}$ pour $i = 0, 1, 2, \ldots, n$ de la matrice $\mathcal{M}$. Cet ensemble de matrices sert à calculer la position relative des objets par rapport au repère de rotation de chaque articulation.

7.4 Le codage et décodage des trajectoires

7.4.1 Le codage des domaines de recherche de $SEARCH$ et $EXPLORE$

Dans le chapitre 3, nous avons vu que les espaces de recherche des algorithmes $SEARCH$ et $EXPLORE$ sont $I\!R^{n*k}$ et $[1, 2, \ldots, t] \times I\!R^{n*m}$ respectivement, où n est le nombre de degrés de liberté du robot, k l'ordre des chemins Manhattan utilisés par $SEARCH$, m l'ordre des chemins Manhattan utilisés par $EXPLORE$ et t l'indice de l'itération de l'algorithme $EXPLORE$. Pour optimiser les fonctions de $SEARCH$ et $EXPLORE$ avec un algorithme génétique, nous devons coder ces espaces en chaînes de bits.

Codage de l'espace de recherche de $SEARCH$

Dans l'algorithme $SEARCH$, nous avons utilisé des chemins Manhattan d'ordre deux. Ainsi, étant donné que le robot a six degrés de liberté, le codage d'une trajectoire est composée de $6 * 2 = 12$ segments Δ_i^j, c'est-à-dire qu'une trajectoire a la forme :

$$\hat{\gamma} = (\Delta_1^1, \Delta_2^1, \ldots, \Delta_6^1, \ldots, \Delta_1^2, \ldots, \Delta_6^2) \tag{7.20}$$

Nous utilisons 9 bits pour coder chacun des ces segments. Par conséquent, un individu de l'algorithme génétique est composé de 12 segments S_i^j de 9 bits chacun, soit 108 bits par individu :

$$(S_1^1, S_2^1, \ldots, S_6^1, \ldots, S_1^2, \ldots, S_6^2)$$

Nous notons par $Bin(S)$ la fonction de transformation de valeurs binaires en valeurs décimales et $\hat{q}_\circ = (\theta_1, \theta_2, \ldots, \theta_i, \ldots \theta_6)$ la configuration initiale. Ainsi, soit B_Max_i et B_Min_i les butés mécaniques du degré de liberté i du robot et étant donné que pour chaque segment S_i^j il existe 512 valeurs possibles, Δ_i^j est décodée de la manière suivante :

$$\Delta_i^j = \frac{(B\_Max - B\_Min) * Bin(S_i^j)}{512} - \theta_i \qquad (7.21)$$

De cette façon, toutes les valeurs possibles de déplacement pour chacun des degrés de liberté sont codées par les 9 bits composant un segment. Après ce décodage, on applique l'opération de rebondissement décrite dans le chapitre 4.

Codage de l'espace de recherche d'$EXPLORE$

Le codage des trajectoires utilisées par l'algorithme $EXPLORE$ est semblable à celui de l'algorithme $SEARCH$. Nous avons utilisé des trajectoires Manhattan d'ordre deux. Ainsi, une trajectoire est codée d'une manière similaire à l'expression 7.20. La différence est le besoin de coder le point de départ de la trajectoire correspondant à une des balises. Une trajectoire est alors de la forme :

$$(L_k, \Delta_1^1, \Delta_2^1, \ldots, \Delta_6^1, \ldots, \Delta_1^2, \ldots, \Delta_6^2) \qquad (7.22)$$

où $L_k = (\theta_{k,1}, \theta_{k,2}, \ldots, \theta_{k,6})$ est la balise de départ. Nous utilisons 9 bits pour coder l'indice de la balise de départ et 9 bits pour les segments Δ_i^j soit 117 bits par individu. Un individu de l'algorithme génétique de $EXPLORE$ a la forme suivante :

$$(SL, S_1^1, S_2^1, \ldots, S_6^1, \ldots, S_1^2, \ldots, S_6^2)$$

où SL est le vecteur codant la balise L_k de départ, c'est-à-dire : soit t la $t-$ième itération de l'algorithme $EXPLORE$ nous avons $k = (\lfloor t * Bin(SL)/512 \rfloor + 1)$. Nous obtenons alors :

$$\Delta_i^j = \frac{(B\_Max - B\_Min) * Bin(S_i^j)}{512} - \theta_{k,i}$$

7.4.2 Le décodage d'une trajectoire

Dans ce paragraphe, nous décrivons les processus exécutés pour l'obtention d'une trajectoire à partir du codage tout en utilisant une technique de rebondissement particulière. Dans le paragraphe 4.2.2 nous avons introduit la technique de rebondissement grâce à l'intervalle $A_m = [x_i^{min}, x_i^{max}] \in \mathcal{C}_{libre}^{\epsilon}$ défini à partir d'une configuration q_{m-1} et d'un axe x_i de mouvement (voir figure 4.4, page 72). Le calcul de l'intervalle A_m implique deux calculs de débattements: un pour l'obtention de x_i^{min} et l'autre pour x_i^{max}. Comme nous l'avons déjà vu dans le chapitre 4, q_{m-1} est une configuration intermédiaire entre x_i^{min} et x_i^{max}. Ainsi nous pouvons définir une autre sémantique de rebondissement en utilisant la technique suivante :

si la valeur Δ_i^j est positive ou égale à 0, alors le rebondissement est calculé dans l'intervalle $[x_i^{m-1}, x_i^{max}]$ sinon, il est calculé dans l'intervalle $[x_i^{min}, x_i^{m-1}]$.

L'utilisation de tels intervalles a un net avantage : elle réduit le temps de calcul car on effectue un seul calcul de débattement. Il est possible de voir que l'obtention de l'intervalle A_m implique deux calculs de débattements : $\Gamma(X,Y)$ pour obtenir x_i^{max} et $\Gamma^{-1}(X,Y)$ pour x_i^{min}.

A l'aide du rebondissement précédemment décrit, le processus de décodage transforme la chaîne $\hat{\gamma}$ dans un ensemble ordonné de configurations intermédiaires décrivant la trajectoire, c'est-à-dire $Q(\hat{\gamma}) = (\hat{q}_0, \hat{q}_1, \hat{q}_2, \ldots, \hat{q}_{n*k})$ (voir définition 4.3 page 71). Avant de décrire l'algorithme de décodage utilisé pour obtenir l'ensemble $Q_R(\hat{\gamma})$, nous allons définir le prédicat $\Psi(X,Y,\Delta_i^j,\theta)$. Ce prédicat est défini de la manière suivante :

$$\Psi(X,Y,\Delta_i^j,\theta) = \begin{cases} vrai & \text{si } \ \Delta_i^j \geq 0 \wedge \Gamma(X,Y) \geq \Delta_i^j \\ vrai & \text{si } \ \Delta_i^j < 0 \wedge \Gamma^{-1}(X,Y) \geq |\Delta_i^j| \\ faux & \text{autrement} \end{cases} \tag{7.23}$$

Lorsque $\Psi(X,Y,\Delta_i^j,\theta)$ est déterminé comme vrai, la variable θ est unifiée avec la valeur de Δ_i^j. Sinon, θ est unifiée sous le critère suivant :

$$\theta = \begin{cases} \Gamma(X,Y) & \text{pour } \ \Delta_i^j \geq 0 \\ -\Gamma^{-1}(X,Y) & \text{pour } \ \Delta_i^j < 0 \end{cases}$$

En résumé le prédicat $\Psi(X,Y,\Delta_i^j,\theta)$ est vrai si et seulement si le mouvement circulaire en Δ_i^j de X avec Y fixe ne cause pas une collision, autrement elle nous donne le débattement maximal θ. On peut remarquer que si $\Psi(X,Y,\Delta_i^j,\theta) = faux$ la valeur de θ est en réalité x_i^{min} ou x_i^{max} selon la valeur de Δ_i^j.

L'intérêt de l'utilisation du prédicat Ψ se trouve dans les conditions $\Gamma(X,Y) \geq \Delta_i^j$ et $\Gamma^{-1}(X,Y) \geq |\Delta_i^j|$ de la formulation 7.23. Grâce au modèle sphérique, ces deux formules peuvent être déterminées comme vraies sans la réalisation de tous les calculs requis par $\Gamma(X,Y)$ (voir la formulation 7.17). Par exemple, la détermination de la valeur de vérité de $\Gamma(X,Y) \geq \Delta_i^j$ implique les calculs suivants :

$$\forall x \in X, \forall y \in Y \quad \psi(x,y) \geq \Delta_i^j$$

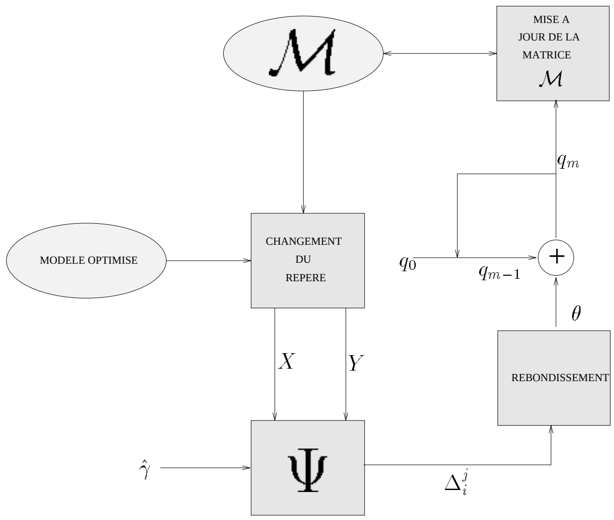


FIG. 7.13 - Processus de décodage d'une trajectoire.

Etant donné deux objets $x \in X$, et $y \in Y$ le modèle sphérique construit par le prédicat Ψ peut répondre dans certaines conditions si la proposition $\psi(x, y) \geq \Delta_i^j$ est vraie. Ainsi, le débattement $\psi(x, y)$ n'est pas nécessairement calculé ce qui revient à réduire le temps de calcul.

Nous pouvons décrire maintenant le processus de décodage. Etant donné la configuration courante q_{m-1} et Δ_i^j (où $m = n*(j-1)+i$) nous pouvons calculer q_m (voir page 71). Premièrement, on exprime l'ensemble X des objets mobiles et l'ensemble Y des objets statiques dans le repère R_i. Après, on applique le prédicat $\Psi(X, Y, \Delta_i^j, \theta)$ en obtenant la valeur θ. On utilise alors θ , Δ_i^j et q_{m-1} dans la fonction de rebondissement pour obtenir q_m. Finalement, la matrice $\mathcal{M}$ est actualisée. Cette boucle continue jusqu'à ce que l'on arrive à la fin de la trajectoire. Le processus de décodage est montré dans la figure 7.13.

7.4.3 Reconstruction de la trajectoire finale

De même que pour le robot mobile holonome, la trajectoire finale est construite grâce à l'arbre de balises représenté par un tableau. Un exemple de cette structure est montré dans le tableau 7.1. Ce tableau représente l'arbre de balises de les figures 7.14 à 7.17 .

Avant d'exécuter la trajectoire finale, nous utilisons l'algorithme décrit ci-après pour la lisser. Cet algorithme utilise l'ensemble propre d'une trajectoire Manhattan défini dans la section 4.2.1.

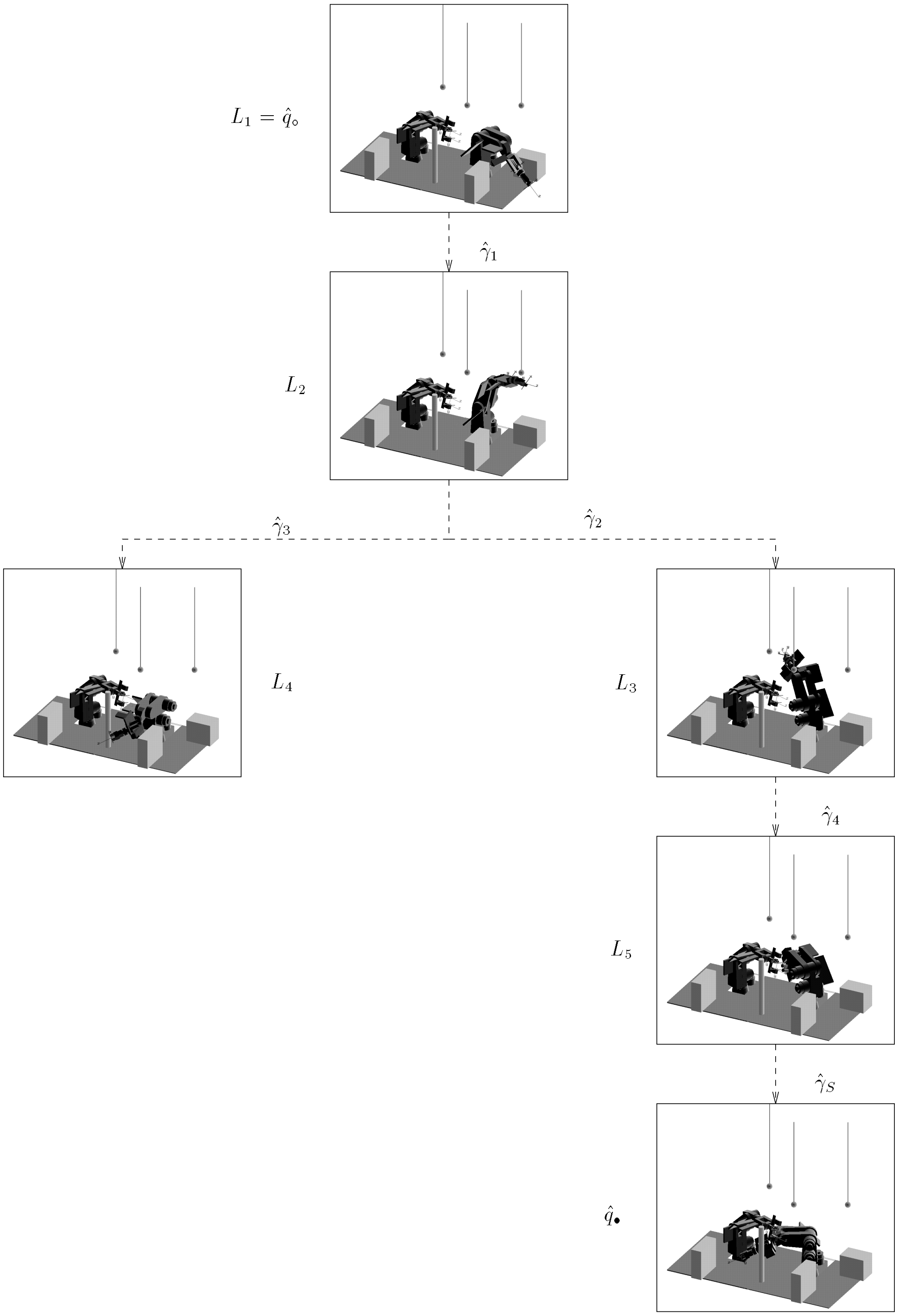


FIG. 7.14 - L'arbre de balises.

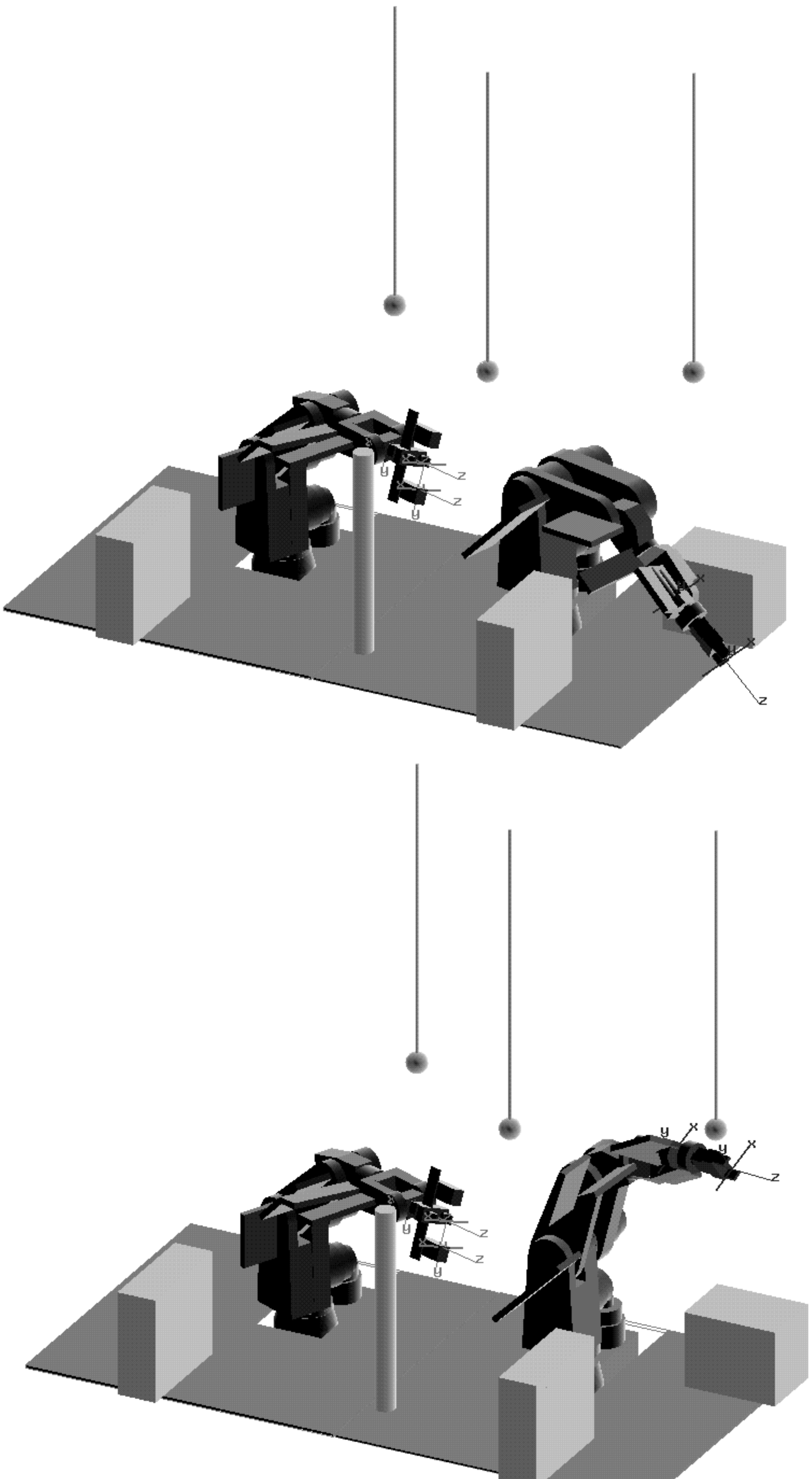


FIG. 7.15 - Balises L_1 et L_2.

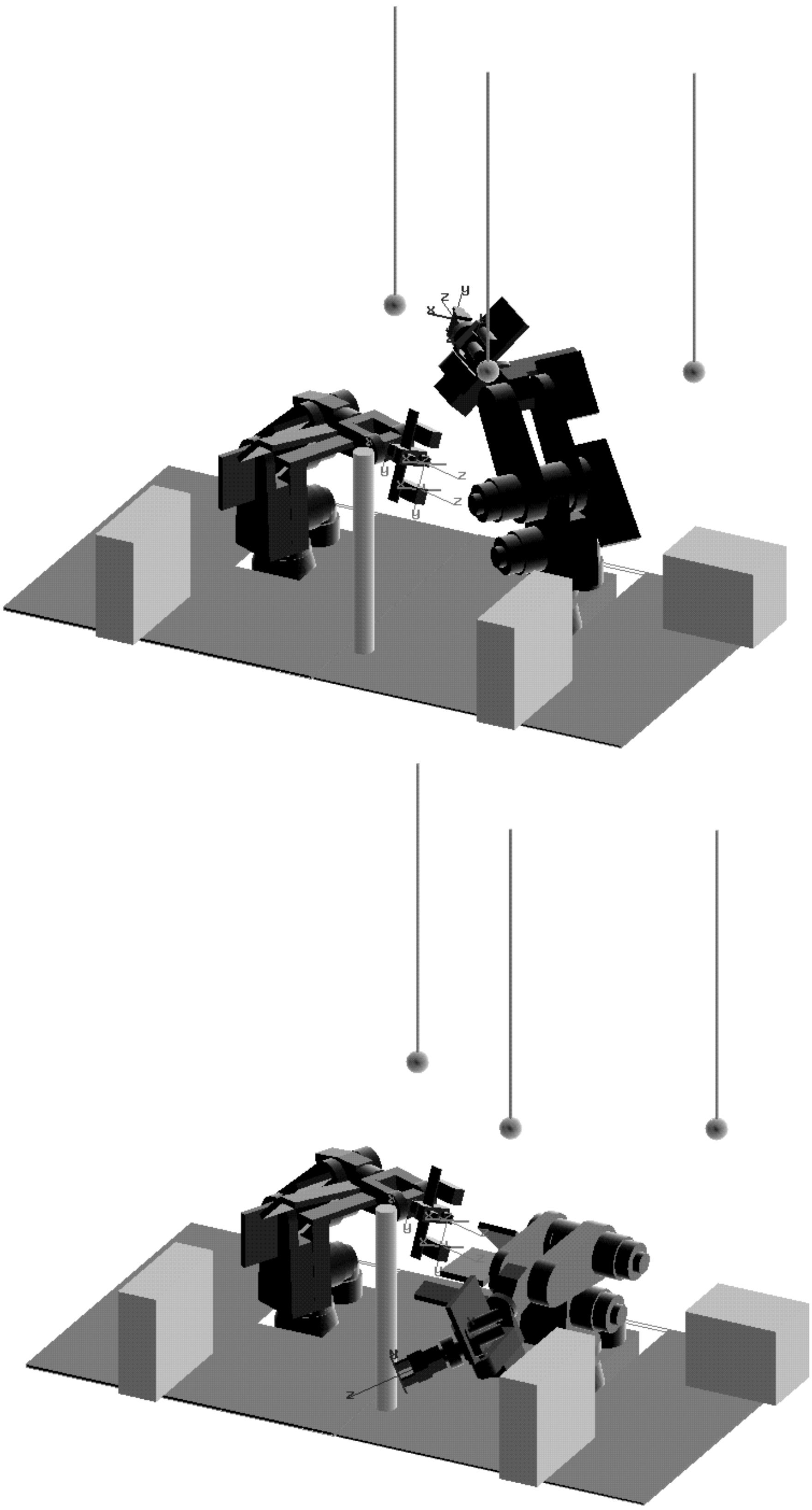


FIG. 7.16 - Balises L_3 et L_4.

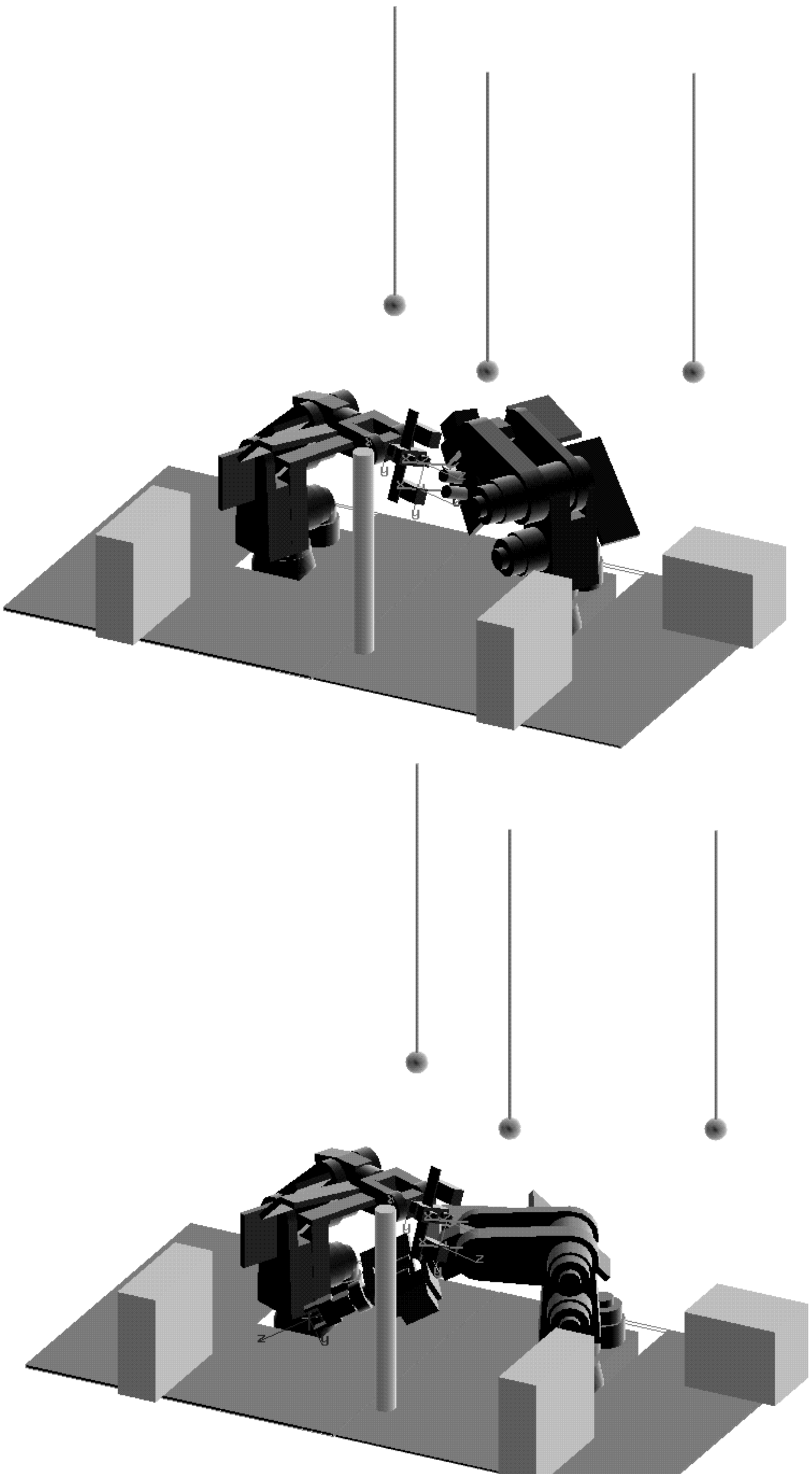


FIG. 7.17 - Balises L_5 et $\hat{q}_{\bullet}$.

i	L_i	**Père de** L_i	$\hat{\gamma}_i$
1	L_1	nul	nul
2	L_2	1	$\hat{\gamma}_1$
3	L_3	2	$\hat{\gamma}_2$
4	L_4	2	$\hat{\gamma}_3$
5	L_5	3	$\hat{\gamma}_4$

TAB. 7.1 - Tableau représentant l'arbre de balises

Une *simplification* d'un chemin Manhattan $\hat{\gamma}$ d'ordre l dans un autre chemin Manhattan $\hat{\gamma}_a$ d'ordre $k < l$ existe, si et seulement s'il existe un chemin Manhattan $\hat{\gamma}_b$ d'ordre 1 tel que $\hat{\gamma}_b(0) = q_i$ et $\hat{\gamma}_b(1) = q_j$ pour $q_i, q_j \in Q(\hat{\gamma})$, $i = 0, \ldots, (l-2)$ et $j = (i+2), \ldots, l$. Une manière de simplifier une trajectoire Manhattan est alors la suivante: soit $G(QM(\hat{\gamma})) = \{(i,j) | i = 0, \ldots (l-2), (i+2) \leq j \leq l, MP(\hat{q}_i, \hat{q}_j)\}$ et $(i', j') \in G(QM(\hat{\gamma}))$: $\forall (i,j) \in G(QM(\hat{\gamma})), (j' - i') \geq (j - i)$, la fonction $Simp(QM(\hat{\gamma}))$ est définie de la manière suivante:

$$Simp(QM(\hat{\gamma})) = \{q_0, \ldots, q_{i'}\} \cup \{q_{j'}, q_{j'+1}, \ldots, q_l\}$$

Nous pouvons alors définir un algorithme très simple pour le lissage de trajectoires Manhattan:

$Lissage(\hat{\gamma})$
begin
  $B_0 = QM(\hat{\gamma})$;
  $i = 0$;
  do
    $B_{i+1} = Simp(B_0)$;
  i = i+1;
  while($B_i = B_{i-1}$);
  return($QM^{-1}(B_0)$);
end

Pour expliquer cet algorithme, nous utilisons la figure 7.18 qui montre la procédure de simplification d'une trajectoire. Nous avons représenté par des segments de droite le passage d'une configuration $\hat{q}_{m-1}$ à une autre $\hat{q}_m$, les obstacles sont représentés par des ellipses. Ainsi, en utilisant cette représentation, le prédicat $MP(\hat{q}_i, \hat{q}_j)$ est vrai si et seulement si le segment de droite $\overline{\hat{q}_i \hat{q}_j}$ ne coupe pas une des ellipses.

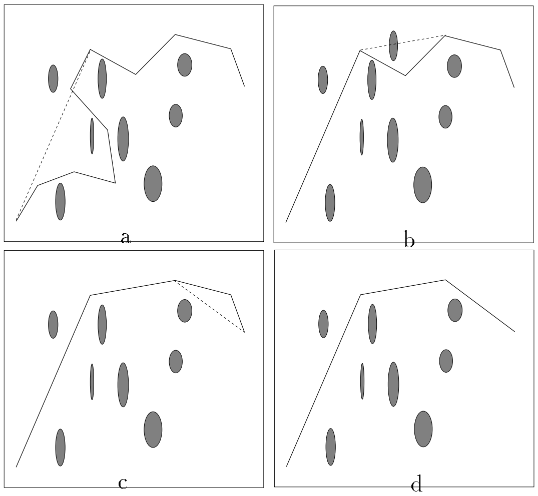


FIG. 7.18 - Simplification.

7.5 Discussion

7.5.1 La difficulté de trouver une solution

Les expérimentations réalisées nous confirment que la difficulté d'un problème de planification n'est pas tout à fait reliée au nombre d'obstacles ou au nombre de degrés de liberté du robot. La difficulté de résoudre un problème de planification est plutôt réliée au nombre de "pièces" de l'espace des configurations et à la taille des passages permettant de "déambuler" dans l'espace libre.

L'algorithme Fil d'Ariane a montré une vraie adaptation à la complexité des problèmes posés. Entre autres expérimentations, nous avons réalisé plusieurs expérimentations, en utilisant des environnements et des robots fictifs. Nous avons créé des situations avec des minima locaux et des situations où il fallait passer par un endroit bien précis pour trouver la solution. L'algorithme Fil d'Ariane a trouvé à chaque fois une trajectoire. Les problèmes les plus difficiles ont été les problèmes où il fallait partir d'une configuration initiale placée dans un voisinage complètement libre d'obstacles et dont la configuration finale se trouvait après un passage étroit et d'accès unique. Les problèmes les plus faciles ont été ceux correspondant à la situation contraire. Si nous plaçons la configuration initiale dans une région entourée d'obstacles, presque toutes les trajectoires engendrées font sortir immédiatement le robot de cette région.

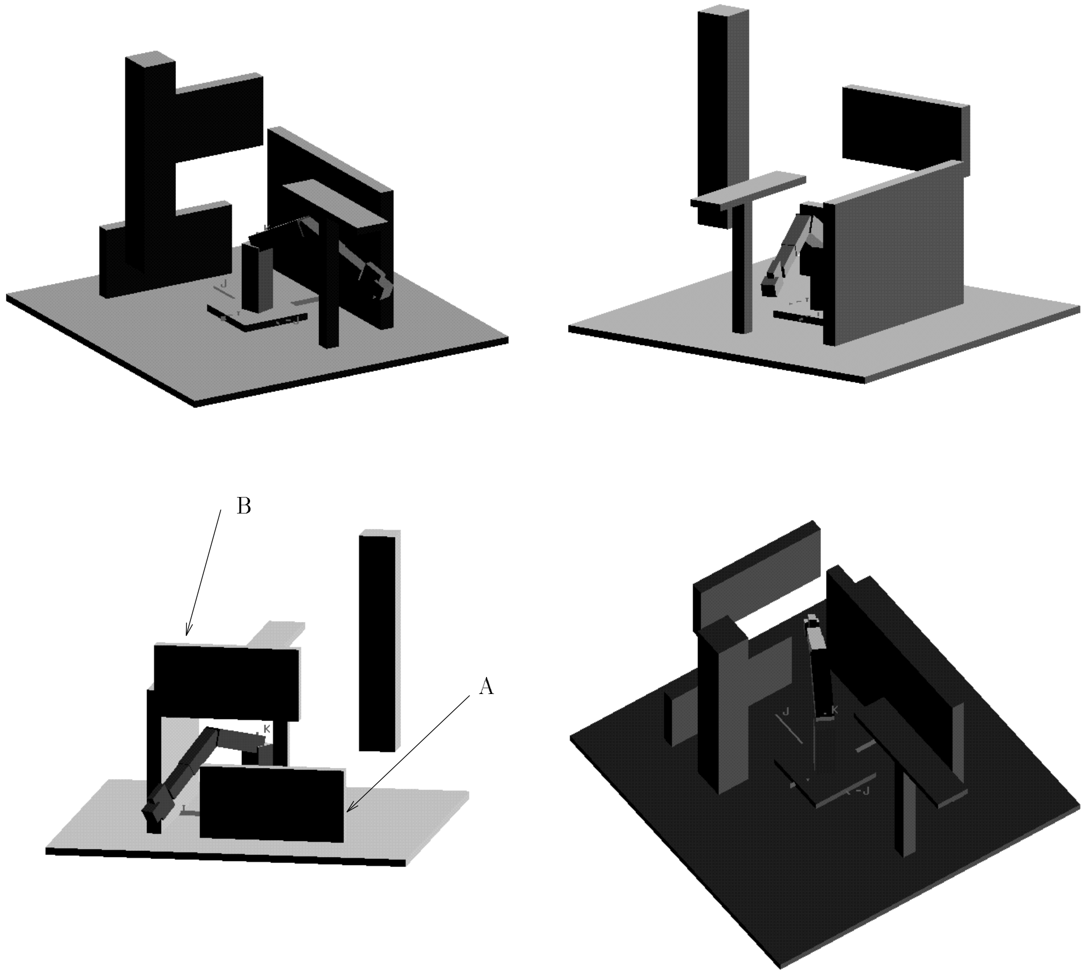


FIG. 7.19 - Un exemple de planification fictif.

La figure 7.19 montre un environnement et un robot fictif pour lesquels nous avons planifié des trajectoires. Pour donner une meilleur idée de la composition de l'environnement, deux vues de la configuration initiale et deux vues de la configuration finale sont montrées respectivement. La seule solution à ce problème est de passer entre les parallélépipèdes similaires A et B. Dans cet exemple quatorze balises sont nécessaires pour calculer la trajectoire finale. Lorsque nous changeons la configuration initiale en configuration finale et vice versa, le nombre de balises nécessaires pour calculer la trajectoire finale diminue considérablement : seules cinq balises sont requises.

7.5.2 Les calculs requis pour une trajectoire Manhattan et les calculs filtrés

Les calculs requis

Le coût d'évaluation d'une trajectoire dépend directement du nombre d'obstacles, du nombre de degrés de liberté et de l'ordre des trajectoires. Regardons tout d'abord la complexité d'une trajectoire Manhattan d'ordre 1.

Lorsqu'on évalue une trajectoire Manhattan, on modifie séquentiellement les différents degrés de liberté. Une variation de la première articulation provoque le mouvement de toutes les composantes du robot. Une variation de la deuxième articulation provoque le mouvement des composantes placées après cette articulation; seules les composantes de la première articulation restent statiques. Pour un robot à n degrés de liberté, lorsqu'on bouge l'articulation i, les composantes de l'articulation $\{1, 2, \ldots, i-1\}$ restent statiques et les composantes de l'articulation $\{i, i+1, \ldots, n\}$ se trouvent en mouvement. Par conséquent, si k est le nombre d'obstacles dans la scène, le mouvement de la première articulation requiert $(n * k)$ calculs de débattements entre parallélépipèdes (si on considère un parallélépipède par articulation). Pour la deuxième articulation $((n-1) * k)$ calculs sont requis, pour la troisième $((n-2) * k)$, etc. Cela nous amène à réaliser $(\frac{n^2+n}{2} * k)$ calculs de débattements entre parallélépipèdes pour une trajectoire Manhattan d'ordre 1.

Soit ℓ l'ordre des trajectoires Manhattan utilisées; le nombre de calculs à réaliser est :

$$\frac{n^2+n}{2} * k * \ell$$

Le nombre de calculs élémentaires (contacts de type A, B et C) peut être évalué. Le calcul du débattement entre deux parallélépipèdes, requiere 48 calculs de type point-face, 48 de type face-point et 144 de type arête-arête. Par conséquent, le nombre de calculs élémentaires est :

$$120 * \frac{n^2+n}{2} * k * \ell$$

7.5.3 Le filtrage des calculs

L'expérience nous montre que l'utilisation du modèle sphérique ainsi que d'autres techniques de filtrage réduit très considérablement le nombre de contacts à vérifier et par conséquent le temps de calcul. Le pourcentage du nombre de calculs filtrés par le modèle sphérique est d'environ 85 %, c'est-à-dire que seul 15% des calculs $\psi(x, y)$ doivent être vérifiés. Ainsi, seulement 15 % des parallélépipèdes seront décomposés en entités géométriques pour réaliser les calculs point-face, face-point et arête-arête. Enfin, 65 % en moyenne des calculs $\psi_A(s, f)$ et $\psi_B(f, s)$ (point-face et face-point) ainsi que 90 % en moyenne des calculs $\psi_C(a, a)$ (arête-arête) sont aussi filtrés. En résumé, de tous les contacts possibles intervenant dans l'évaluation d'une trajectoire Manhattan, seuls 3 % en moyenne doivent être calculés.

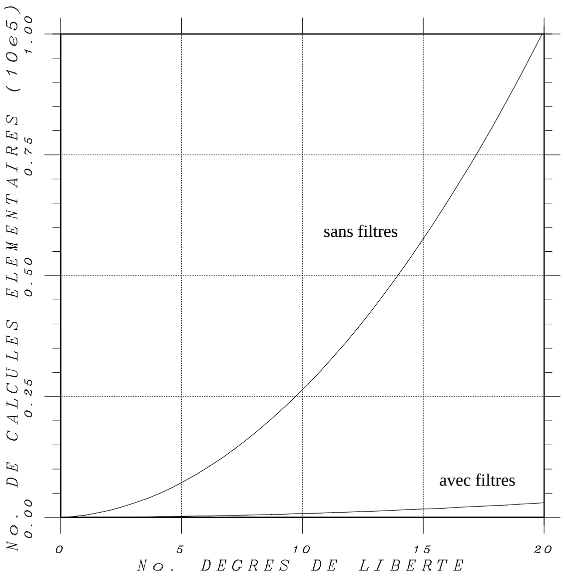


FIG. 7.20 - Nombre de calculs élémentaires requis pour une trajectoire Manhattan d'ordre 1.

La figure 7.20 montre le nombre de calculs élémentaires requis pour chaque objet placé dans l'espace atteignable du robot. Ces résultats ont été calculés pour des trajectoires Manhattan d'ordre 1.

Il est important de noter que malgré l'utilisation de filtres, la réduction du temps de calcul ne se fait pas dans la même proportion que celle concernant la réduction du nombre de calculs. Bien entendu, l'utilisation des filtres a un prix en temps de calcul. L'utilisation des filtres diminue le temps de calcul d'environ 15% du temps total requis sans filtre.

Chapitre 8

Implantation parallèle de l'algorithme Fil d'Ariane

Les fonctions d'évaluation des algorithmes SEARCH et EXPLORE, sont basées sur le calcul des débattements entre les parallélépipèdes du modèle optimisé. Lorsqu'une trajectoire Manhattan d'ordre ℓ est évaluée un nombre considérable de calculs mathématiques est effectué. Dans l'exemple de la figure 7.4 chacun des robots est représenté (dans sa forme la plus simple) par 7 parallélépipèdes, c'est-à-dire un parallélépipède par degré de liberté plus un autre pour représenter la base du robot. De même, les obstacles sont représentés chacun par un parallélépipède. Si nous considérons le robot de gauche comme un obstacle et celui de droite comme mobile nous avons un total de 18 obstacles. Ainsi une trajectoire Manhattan d'ordre 1 pour le robot mobile requiert $\frac{6^2+6}{2} * 18 = 378$ calculs de débattement c'est-à-dire 378 exécutions de la fonction $\psi(x, y)$. Chacun des appels de la fonction $\psi(x, y)$ requiert 240 calculs élémentaires de contact (point-face, face-point, arête-arête). Ainsi, si nous considérons une trajectoire Manhattan d'ordre 5 nous obtenons 378*240*5= 417.600 calculs élémentaires. Même si le modèle sphérique élimine une grande partie de ces calculs, seule l'utilisation d'une machine parallèle permet une réduction du temps de calcul compatible avec nos objectif “temps réel”.

Dans ce chapitre nous présentons la parallélisation de l'Algorithme Fil d'Ariane pour un bras manipulateur à six degrés de liberté. Bien qu'ayant implanté plusieurs versions parallèles de l'algorithme, nous décrirons uniquement la plus efficace. Dans un premier temps, nous présenterons la parallélisation logique avec ses différents et les processus concernés. Puis, nous décrirons la parallélisation physique en indiquant la configuration de la machine parallèle et le placement des processus.

8.1 Les Niveaux de parallélisation

L'algorithme Fil d'Ariane, les algorithmes génétiques et la fonction d'évaluation sont "par nature" parallélisables.

1. Dans l'algorithme File d'Ariane, SEARCH et EXPLORE peuvent travailler de façon indépendante. Pendant que SEARCH cherche une solution EXPLORE produit la balise suivante.
2. L'évaluation des individus, dans les algorithmes génétiques, peut se faire en parallèle.
3. Les fonctions d'évaluation, basées principalement sur l'utilisation de la fonction $\psi(x, y)$, sont aussi parallélisables. La fonction $\psi(x, y)$ est en effet décomposable en trois sous-fonctions indépendantes $\psi_A(sm, fs), \psi_B(fm, ss)$ et $\psi_C(am, as)$.

Dans la suite nous présenterons les différents processus intervenant dans les trois niveaux de parallélisation.

8.1.1 Le premier niveau de parallélisation

A ce niveau, nous avons trois processus : le processus **"Maître"**, puis **"l'algorithme génétique Search"** et finalement **"l'algorithme génétique Explore"**, ces deux derniers seront notés simplement comme "Search" et "Explore". [1]

Le rôle du processus "Maître" est de gérer et de coordonner les processus "Search" et "Explore" qui représentent eux même deux algorithmes génétiques parallèles (**PGA**). Premièrement le processus "Maître" envoie le modèle optimisé au processus "Search" et "Explore". Ce modèle est utilisé par les fonctions d'optimisation respectives. Après l'exécution d'un nombre prédéterminé de générations des algorithmes génétiques "Search" et "Explore", le processus "Maître" reçoit la population

[1] Même s'il s'agit indirectement de l'exécution des algorithmes SEARCH et EXPLORE nous préférons faire une différence entre l'exécution des algorithmes génétiques et des algorithmes proprement dits.

finale de l'algorithme génétique "Search" avec l'objectif de sélectionner le meilleur individu. Si la trajectoire codée par l'individu n'engendre pas un chemin à la configuration finale désirée, le processus "Maître" reçoit la population de l'algorithme génétique de "Explore" et sélectionne le meilleur individu, cet individu code la nouvelle balise qui est diffusée comme telle au processus "Explore" et au processus "Search" comme nouveau point de départ. Ensuite les algorithmes génétiques "Search" et "Explore" sont réexécutés jusqu'à ce que "Search" trouve un individu qui code un chemin de la balise courante au but.

La description en pseudo-Occam des processus "Maître", "Search" et "Explore" est la suivante :

```
maître(modèle_optimisé)
begin
    SEQ
        L_1 := q̂_∘;
        EL := {L_1};
            PAR
                envoyer(Search,modèle_optimisé); /* envoi du modèle optimisé */
                envoyer(Explore,modèle_optimisé);
        i := 1
        while( L_i ≠ q̂_• ) do
            SEQ
                PAR
                    envoyer(Search,L_i); /* envoi de la balise i */
                    envoyer(Explore,L_i);
                recevoir(Search,population_Search)
                meilleur_search = min(population_Search)
                if (meilleur_Search.value = 0) /* une trajectoire à q̂_• a été trouvée */
                    i := i+1;
                    L_i := E(meilleur_Search.trajectoire) /* extrémité du chemin */
                else
                    SEQ
                        recevoir(Explore,population_Explore)
                        meilleur_explore = max(population_Explore)
                        i := i + 1;
                        L_i := E(meilleur_explore.trajectoire) /* extremité du chemin */
end
```

```
Search()
begin
    SEQ
        recevoir(Maître,modèle_optimisé);
        while(¬stop)
            recevoir(Maître,q̂₀);
            PGA(modèle_optimisé);
            envoyer(Maître,population);
end
```

```
Explore()
begin
    SEQ
        i:=1;
        recevoir(Maître,modèle_optimisé);
        while(¬stop)
            recevoir(Maître,L_i);
            PGA(modèle_optimisé);
            envoyer(Maître,population);
            i:= i+1;
end
```

8.1.2 Le deuxième niveau de parallélisation

"Search" et "Explore" contiennent tous les deux un algorithme génétique parallèle. Comme nous l'avons présenté dans les chapitres précédents le principe de base est d'évaluer p individus placés sur p processus, c'est-à-dire un individu par processus. Ensuite, chaque processus envoie son individu à ses voisins (Ces voisins sont prédéfinis), puis il sélectionne le meilleur individu parmi ses voisins pour exécuter les opérations de cross-over et mutation. Un nouvel individu est ainsi obtenu dans chacun des processus, si cet individu est meilleur que son prédécesseur il le remplace. Cet algorithme est exécuté "ng" fois où "ng" est un nombre déterminé par l'utilisateur. A chaque génération on ne cherche pas la meilleure solution présente dans les différents processus, le coût en serait trop important. La sélection du meilleur individu de la population sera seulement effectuée à la fin du nombre total de générations.

Les processus en pseudo-Occam utilisés par l'algorithme génétique parallèle sont les suivants :

```
AGi(modèle_optimisé)
begin
    SEQ
        créer(individu)
        évaluer(invividu)
        for i = 1 to ng /* nombre de générations */
            SEQ
                for j = 1 to nombre_voisins
                    PAR
                        envoyer(voisin_j, individu);
                for j = 1 to nombre_voisins
                    PAR
                        recevoir(voisin_j, candidats_j);
                meilleur = sélection_meilleur(candidats);
                nouvel_individu = cross-over(individu, meilleur);
                nouvel_individu = mutation(nouvel_individu);
                évaluer(nouvel_individu,modèle_optimisé );
                if(nouvel_individu.value < individu.value)
                    individu = nouvel_individu;
end
```

```
PAG(modèle_optimisé)
begin
    SEQ
        for i = 1 to nombre_individus
            PAR
                AGi(modèle_optimisé);
end
```

8.1.3 Troisième niveau de parallélisation

La parallélisation massive

Nous pourrions imaginer une parallélisation massive en trois niveaux de la fonction d'évaluation utilisée dans les deux algorithmes génétiques. Ces fonctions sont basées principalement dans l'utilisation du prédicat $\Psi(X, Y, \Delta_i^j, \theta)$ (voir chapitre précédent) calculé grâce à la fonction $\Gamma(X, Y)$ où X est l'ensemble des objets mobiles, Y l'ensemble des objets statiques :

$$\Gamma(X, Y) = \min_{x \in X, y \in Y} \psi(x, y)$$

Ainsi, le *premier niveau* serait l'exécution en parallèle de $Card(X \times Y)$ processus où chacun d'entre eux exécute le calcul de $\psi(x, y)$ pour un couple différent de parallélépipèdes $(x, y) \in X \times Y$. Le *deuxième niveau* de parallélisation serait fait à partir de la définition même de $\psi(x, y)$:

$$\psi(x,y) = \min\{\theta_A, \theta_B, \theta_C\}$$

Ce qui revient à créer trois types de processus différents, un pour chaque angle de contact θ_A, θ_B et θ_C (voir page 117). Le *troisième niveau* de parallélisation consisterait ainsi en la décomposition des fonctions $\psi_A(sx, fy), \psi_B(fx, sy)$ et $\psi_C(ax, ay)$ en 48 processus pour le calcul de θ_A, 48 processus pour celui de θ_B et 144 processus[2] pour θ_C.

Une telle parallélisation massive impliquerait un nombre énorme de processus. Le gain théorique en temps de calcul est compensé par la perte de temps due à la création de ces processus, à la transmission du modèle optimisé, à la décomposition des entités géométrique, etc. De plus la difficulté de placement des processus sur les processeurs croit avec le nombre de processus [83]. Nous avons opté pour une architecture plus simple mais plus efficace avec trois niveaux de parallélisation qui fait l'objet du paragraphe suivant.

La parallélisation réalisée

Dans la parallélisation réalisée, le premier niveau est substitué par le calcul séquentiel de $\Gamma(X,Y)$. L'ensemble des calculs équivalents sont effectués en éclatant l'ensemble des calculs de contacts dans différents processus placés dans le deuxième niveau de parallélisation de la fonction d'évaluation.

La parallélisation du prédicat $\Gamma(X,Y)$ contient deux niveaux : dans le premier, nous avons le processus **"Gamma"** et les processus **"Psi_AB"** et **"Psi_C"**. Le processus Gamma représente le calcul de $\Gamma(X,Y)$ et est effectué grâce au processus "Psi_AB" calculant deux angles de contact Θ_A et Θ_B, et Psi_C l'angle de contact Θ_C. Ces angles sont définis comme suit :

$$\Theta_A = min\{\theta_{A_{i,j}} |\ \theta_{A_{i,j}} = \min_{sx \in Sx_i, fy \in Fy_j} \psi_A(sx, fy) \text{ pour } Sx_i \in x_i \in X \wedge Fy_j \in y_j \in Y\}$$

$$\Theta_B = min\{\theta_{B_{i,j}} |\ \theta_{B_{i,j}} = \min_{fx \in Fx_i, sy \in Sy_j} \psi_B(fx, sy) \text{ pour } Fx_i \in x_i \in X \wedge Sy_j \in y_j \in Y\}$$

$$\Theta_C = min\{\theta_{C_{i,j}} |\ \theta_{C_{i,j}} = \min_{ax \in Ax_i, ay \in Ay_j} \psi_C(ax, ay) \text{ pour } Ax_i \in x_i \in X \wedge Ay_j \in y_j \in Y\}$$

Ainsi, par exemple, Θ_A représente le débattement maximal possible des contacts de type A de l'ensemble X d'objets en mouvement par rapport à l'ensemble Y d'objets statiques. Les calculs de Θ_A et Θ_B sont effectués dans le même processus "Psi_AB" car le temps de calcul nécessaire pour obtenir ces deux valeurs est proche

[2] $Card(Sx \times Fy) = 48$, $Card(Fx \times Sy) = 48$, $Card(Ax \times Ay) = 144$

de celui qui est nécessaire pour obtenir Θ_C. Le processus "Psi_AB" calcule alors $\Theta_{AB} = \min\{\Theta_A, \Theta_B\}$.

Le deuxième niveau de parallélisation des fonctions d'évaluation est effectué dans l'utilisation de dix sous-processus parallèles **"psi_ab"** pour le calcul de Θ_A et Θ_B et dix autres **"psi_c"** pour le calcul de Θ_C. Chacun des processus "psi_ab" effectue alors le calcul des fonctions $\psi_A(sx, fy), \psi_B(fx, sy)$ et chacun des processus "psi_c" le calcul de $\psi_C(ax, ay)$. A chacun des processus "psi_ab' et "psi_c" correspond un sous ensemble de $Sx_i \times Fy_j$, $Fx_i \times Sy_j$ ou $Ax_i \times Ay_j$ où $Sx_i \in x_i \in X$, $Sy_j \in y_j \in Y$, $Fx_i \in x_i \in X$ et $Fy_j \in y_j \in Y$. Dans cette architecture tous les processeurs sont chargés avec le même modèle de l'environnement et la même matrice $\mathcal{M}$ pour exécuter le changement de repère des entités géométriques qui le concernent et la mise à jour du modèle. Une fois finis les calculs correspondants, les processus "psi_ab et "psi_c" envoient la plus petite valeur de débattement au processus "Psi_AB" ou "Psi_C" selon le cas. Ensuite, Gamma obtient le débattement le plus petit parmi les deux valeurs calculées par "Psi_AB" et "Psi_C". Enfin, cette valeur est propagée à l'ensemble des processus du deuxième niveau "psi_ab" et "psi_c" pour la mise à jour des matrices[3] $\mathcal{M}$ et du modèle. La figure 8.1 montre les différents niveaux de parallélisation.

Les processus suivants prennent pour paramètre la valeur Δ_i^j. Ainsi que nous l'avons montré dans le chapitre précédent nous pouvons filtrer les calculs où nous sommes certains que l'angle de contact est supérieur à la valeur Δ_i^j. Les processus en pseudo-Occam sont les suivants :

```
Gamma(φ̂)
begin
SEQ
    for i = 1 to ordre_chemin
        SEQ
            for j = 1 to nombre_degrés_de_liberté
                PAR
                    Θ_AB = Psi_AB(Δ_i^j);
                    Θ_C = Psi_C(Δ_i^j);
            θ = min(Θ_AB, Θ_C);
                return(θ);
end
```

[3] Nous devons nous rappeler que ces processus n'ont pas fait la mise à jour de la matrice $\mathcal{M}$ ni du modèle optimisé car la valeur du débattement $\Gamma(X, Y)$ était inconnue.

```
Psi_AB(θ)
begin
    SEQ
        for i = 1 to 10
            PAR
                θ_{ab_i} = psi_ab_i(θ);
        Θ_{AB} = min(θ_{ab_1}, θ_{ab_2}, ..., θ_{ab_10});
        return(Θ_{AB});
end
```

```
Psi_C(θ)
begin
    SEQ
        for i = 1 to 10
            PAR
                θ_{c_i} = psi_c_i(θ);
        Θ_C = min(θ_{c_1}, θ_{c_2}, ..., θ_{c_10});
        return(Θ_C);
end
```

8.2 L'implantation physique

Nous avons utilisé comme matériel de développement, un Méganode à 128 transputers du type T800 commercialisé par la société Telmat. Le Méganode est constitué de 4 Tnode16/32 configurés avec 32 T800.

Le Tnode 16/32 est une machine multi-processeurs sans mémoire commune composée de transputers T800. Les processeurs sont connectés à travers deux commutateurs programmables. Un des commutateurs est un *"crossbar"* à 72 entrées et 72 sorties, permettant de connecter 32 transputers de travail (TT), deux transputers de service (TS : disque, mémoire, etc) et un transputer de contrôle (TC). L'autre commutateur (*crossbar* C004 à 32 entrées et 32 sorties) permet de relier le TC aux autres transputers et de réaliser l'interface avec la machine hôte. La carte d'interface assure la liaison entre la machine hôte et le Tnode. Un bus de contrôle, transportant quelques signaux de services tels que reset, analyse et erreur, permet au TC de surveiller et de contrôler, l'activité des TT et TS.

Le Méganode est piloté depuis une machine hôte de type SUN 4, grâce à une carte d'interface VOLVOX-4 de la société Archipel. La carte VOLVOX est constituée de 4 T800 avec chacun 4 Mo de mémoire. Le transputer 0 de cette carte est connecté directement au Méganode et est utilisé comme processeur de travail permettant de gérer les Entrées/Sorties clavier, écran, fichiers.

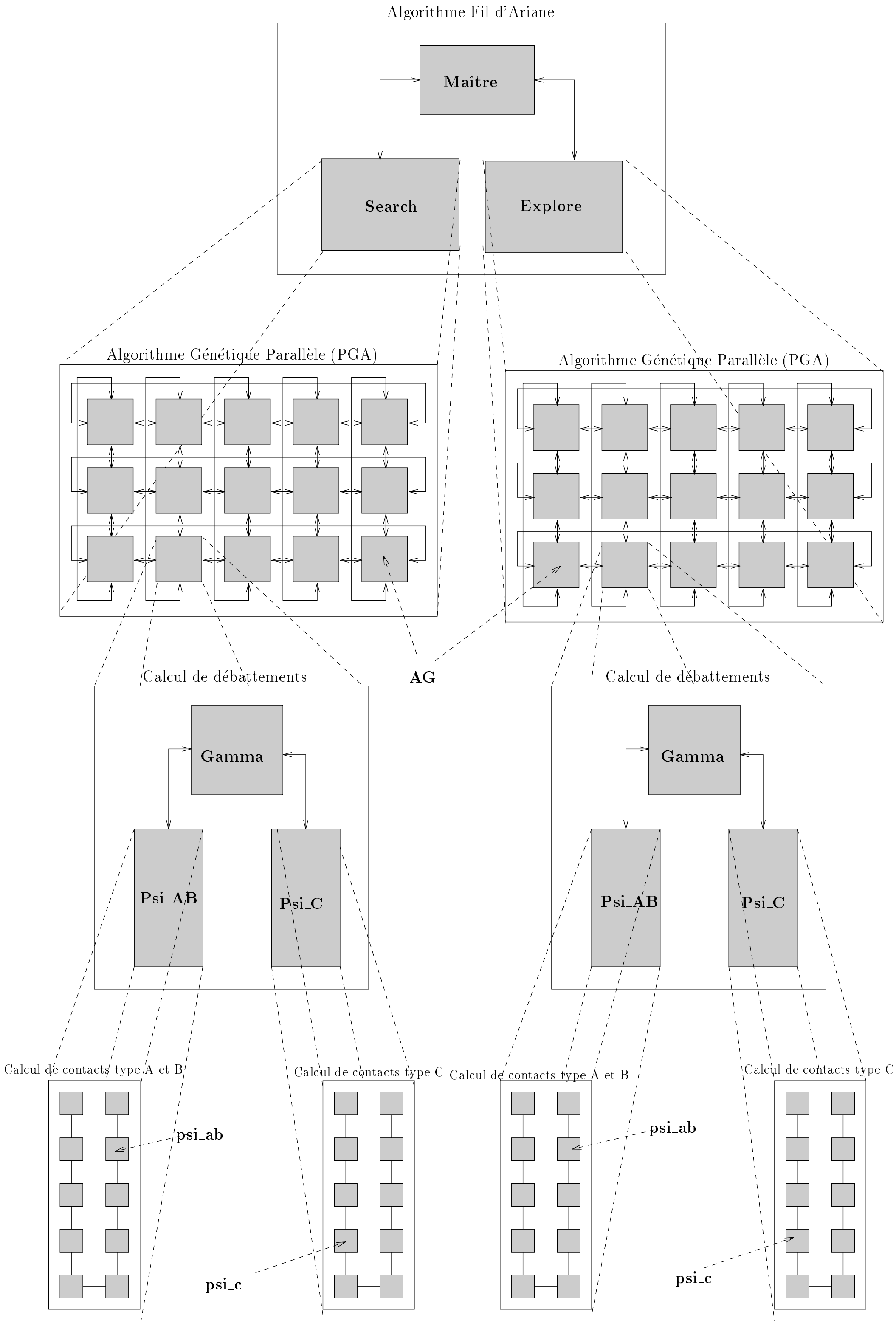


FIG. 8.1 - Niveaux de parallélisation de l'Algorithme Fil d'Ariane.

La figure 8.2 représente la configuration du Méganode choisie pour implanter l'Algorithme Fil d'Ariane. Les numéros dans les carrés indiquent les processeurs et les arcs indiquent les canaux physiques de communication. L'architecture de connexion est décrite "explicitement" au central gérant le graphe de connexion.

8.2.1 Organisation physique du premier et du deuxième niveau de parallélisation

L'organisation physique des deux premiers niveaux de parallélisation est implantée de la façon suivante : le processus "Maître" est placé sur le processeur 0, et chacun des processus, "Search" et "Explore" est placé sur un tore formé par 120 processeurs. Etant donné que "Search" et "Explore" ont la même organisation (tout les deux contiennent des algorithmes génétiques parallèles avec une population de la même taille) et qu'ils s'effectuent dans des temps différents, (d'une manière séquentielle) nous pouvons utiliser les mêmes processeurs, pour former un tore physique et placer les deux tores logiques. Ce tore de processeurs est divisé en six "fermes" de 20 processeurs chacune. Un anneau vertical de six processeurs est formé avec le premier processeur de chaque "ferme" $\{1, 21, 31, \ldots, 101\}$. La tâche de cet anneau est l'exécution de l'algorithme génétique parallèle. Ainsi, chaque processeur de l'anneau exécute le processus "AG" décrit dans le paragraphe précédent en utilisant les voisins nord et sud pour l'opération de cross-over. Par exemple, les voisins du processeur 1 sont les processeurs 21 et 101. Chaque processus "AG" compte alors avec une "ferme" de 20 processeurs pour sa fonction d'évaluation (voir figure 8.2).

8.2.2 Organisation physique du troisième niveau de parallélisation

L'ensemble total des opérations de calcul de débattement qui constitue le "cœur" des fonctions d'évaluation sont menées en parallèle. Chacun des six processus "AG" de l'algorithme génétique parallèle utilise une "ferme" pour évaluer son individu. Le calcul de débattement se fait en éclatant l'ensemble total des calculs de contacts entre les processeurs de la "ferme" divisée en deux groupes de 10 processeurs. Les dix premiers sont utilisés pour exécuter le processus "Psi_AB" et les dix autres "Psi_C" (10 processeurs exécutent "psi_ab" et 10 processeurs exécutent "psi_c"). Par exemple, dans la "ferme" formée par les processeurs du premier au vingtième, les processeurs du premier au dixième exécutent le processus "psi_ab" et les processeurs du onzième au vingtième le processus "psi_c". Les tâches de chacun des processeurs sont alors distribuées grâce à trois tables qui affectent un numéro de processeur (de 1 à 20) à un calcul entre deux entités géometriques. La figure 8.3 montre les conventions de numérotation des entités géometriques d'un parallélépipède. Les figures 8.4, 8.5 et 8.6 indiquent quel processeur se charge du calcul entre deux entités données. Par exemple, le processeur 5 est chargé de calculer le contact de type A (ou *point-face*) entre les points 3 de tous les parallélépipèdes mobiles et les faces 4 de tous les parallélépipèdes fixes. De même le processeur 3 est chargé de calculer le contact de type C entre toutes les arêtes 5 des objets mobiles et les arêtes 4 de tous les objets fixes.

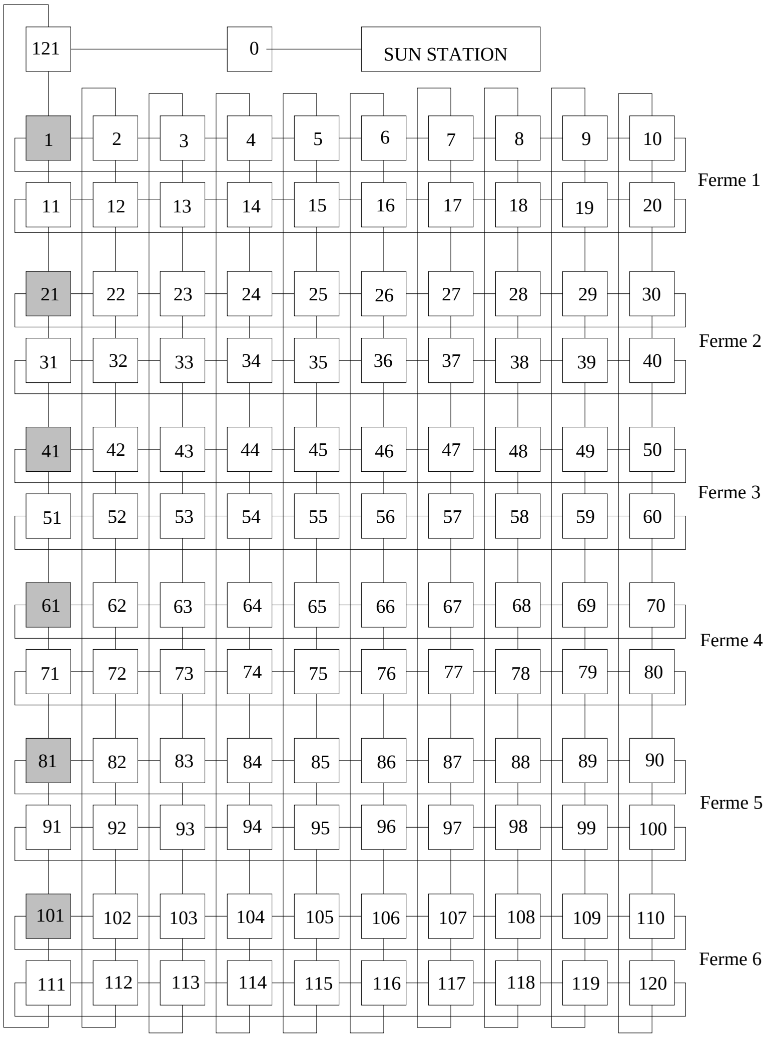


FIG. 8.2 - Configuration du réseau de Transputers.

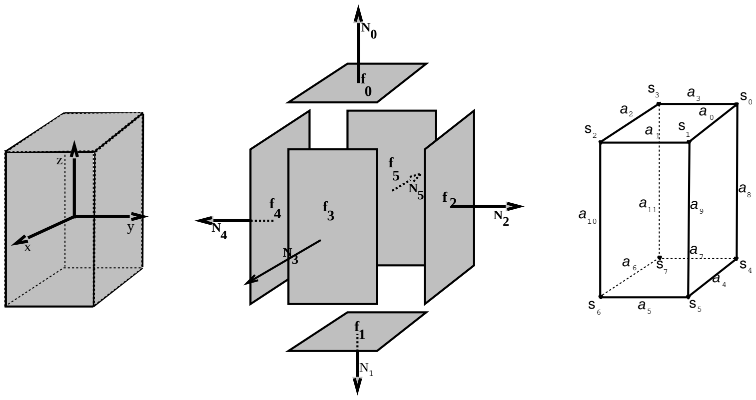


FIG. 8.3 - Décomposition d'un parallélépipède.

POINT \ FACE	0	1	2	3	4	5
0	1	2	3	4	5	6
1	1	2	3	4	5	6
2	1	2	3	4	5	6
3	1	2	3	4	5	6
4	1	2	3	4	5	6
5	7	8	9	9	10	10
6	7	8	9	9	10	10
7	7	8	9	9	10	10
8	7	8	9	9	10	10

FIG. 8.4 - Répartition des calculs de type A sur les processeurs de la "ferme".

FACE \ POINT	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	7	8
1	2	2	2	2	2	2	7	8
2	3	3	3	3	3	3	7	8
4	4	4	4	4	4	4	7	8
4	5	5	5	5	5	5	7	8
5	6	6	6	6	6	6	7	8

FIG. 8.5 - Répartition des calculs de type B sur les processeurs de la “ferme”.

ARETE \ ARETE	0	1	2	3	4	5	6	7	8	9	10	11
0	11	11	11	12	12	12	13	13	13	14	14	14
1	11	11	11	12	12	12	13	13	13	14	14	14
2	11	11	11	12	12	12	13	13	13	14	14	14
3	11	11	11	12	12	12	13	13	13	14	14	14
4	11	11	11	12	12	12	13	13	13	14	14	14
5	15	15	15	16	16	16	17	17	17	18	14	14
6	15	15	15	16	16	16	17	17	17	18	14	14
7	15	15	15	16	16	16	17	17	17	18	14	14
8	15	15	15	16	16	16	17	17	17	18	14	14
9	15	15	15	16	16	16	17	17	17	18	14	14
10	19	19	19	19	19	19	19	19	19	19	19	19
11	20	20	20	20	20	20	20	20	20	20	20	20

FIG. 8.6 - Répartition des calculs de type C sur les processeurs de la “ferme”

Chapitre 9

Perspectives

Dans ce chapitre nous présentons les possibles directions de recherche et les conclusions de cette thèse. Tout d'abord, un ensemble de sujets ouverts reliés à l'algorithme Fil d'Ariane sera présenté. Puis, nous aborderons les conclusions. Dans les conclusions, nous ferons premièrement un rappel des résultats théoriques et expérimentaux obtenus puis, une discussion à-propos de la méthode proposée sera abordée.

9.1 Discussion et direction des recherches

9.1.1 L'algorithme $EXPLORE$ et les diagrammes de Voronoi

L'expression "max min" utilisée par notre méthode est bien connue en géométrie algorithmique [65]. Considérons l'ensemble des balises placées à l'instant t : EL_t. Pour toute balise L_i de EL_t, le lieu géométrique noté $V(i)$ des points de $\mathbb{E}^n$ qui sont plus proches de L_i que de toutes les autres balises est *le polygone de Voronoi* associé à L_i [65].

a b

FIG. 9.1 - Un ensemble de balises et les polygones engendrés.

L'ensemble des régions

$$Vor(EL_t) = \bigcup_{i=1}^{t} V(i)$$

est appelée *le diagramme de Voronoi.* Dans ce diagramme, nous considérons les sommets appartenant à la fermeture de chacun des polygones de Voronoi.

Par exemple, la figure 9.1a montre un espace des configurations pour le robot planaire à deux degrés de liberté dans lequel, dix balises ont été posées par l'algorithme $EXPLORE$. La figure 9.1b montre le diagramme de Voronoi associé aux balises.

Soit C_s l'hypercube le plus petit contenant $\mathcal{C}_{libre}$, il est alors facile de montrer que le point p tel que :

$$p : \max_{p_i \in C_s} \min_{L_i \in EL_t} d(Li, p_i)$$

est soit un sommet du diagramme de Voronoi $Vor(EL_t)$ soit un point de la frontière de C_s. En l'absence d'obstacle, il est donc possible de calculer directement la prochaine balise en construisant le diagramme de Voronoi associé aux balises précédemment posées et en optimisant sur chacun de ses sommets.

Cette première propriété pourrait être utilisée pour "peupler" efficacement les zones libres de l'espace des configurations. En effet, un des problèmes de notre approche est le temps mis à "remplir" ces zones pourtant facile à "explorer".

Maintenant étudions un autre effet de la cellularisation de l'espace des configurations par les polygones de Voronoi associés aux balises.

D'une manière similaire, lorsque l'on cherche q_t tel que :

$$q_t : \max_{q \in \mathcal{P}st(\mathcal{C}_{libre};\ell,EL_t)} d(q, EL_t) = \max_{q \in \mathcal{P}st(\mathcal{C}_{libre};\ell,EL_t)} \min_{L_i \in EL_t} d(L_i, q) \tag{9.24}$$

il existe une fragmentation implicite de la post-image. Les configurations appartenant à la post-image sont regroupées dans des régions; une configuration q appartient à la région $R_{\mathcal{P}}(i)$ de la balise L_i si et seulement si la distance de q à L_i est plus petite que la distance aux autre balises. Le région $R_{\mathcal{P}}(i)$ est définie comme :

$$R_{\mathcal{P}}(i) = \{q \in \mathcal{P}st(\mathcal{C}_{libre}; \ell, EL_t) | d(L_i, q) \leq d(L_j, q) \forall (L_j \neq L_i) \in EL_t\}$$

Par conséquent la configuration la plus éloignée de la région $R_{\mathcal{P}}(i)$ est :

$$Q_i : \max_{q \in R_{\mathcal{P}}(i)} d(L_i, q)$$

où la distance est :

$$D_i = d(L_i, Q_i)$$

Si on réunit tous les points les plus éloignés de chacune des régions $\{R_{\mathcal{P}}(i)\}_{i=1,2,\ldots,t}$ nous pouvons réécrire l'expression 9.24 comme :

$$q_t = Q_i : D_i = \max\{D_1, D_2, \ldots, D_t\} \tag{9.25}$$

Dans ce cas, la convexité ou la non convexité d'une région $R_{\mathcal{P}}$ dépend de la proximité de la balise L_i à la frontière de la post-image. La frontière de la post-image est produite par les $\mathcal{C}$-*obstacles* et par l'ordre des trajectoires utilisées. A mesure que le nombre t de balises augmente le nombre de régions convexes augmente aussi. Cette dernière particularité peut améliorer de façon significative la rapidité de l'algorithme d'optimisation de la fonction $EXPLORE$ car pour une fonction convexe sur un domaine convexe, l'optimum local peut être trouvé facilement [59].

9.1.2 Utilisation de l'information donnée par $EXPLORE(t)$

L'information obtenue par l'algorithme $EXPLORE$ reste un point très intéressant à exploiter. Comme nous l'avons montré, la valeur de $EXPLORE(t)$ peut augmenter par rapport aux valeurs précédentes dès que l'on "découvre" une nouvelle zone libre de l'espace des configurations. Nous proposons deux manières d'utiliser l'information : " zone inexplorée "

1. Il semble logique d'explorer en priorité la région de la post-image engendrée par les "ancêtres" de la balise L_k qui provoque un "saut" de la fonction $EXPLORE$, c'est-à-dire de la balise L_k tel que $EXPLORE(k-1) \ll EXPLORE(k)$. Cela est possible en utilisant l'information de la structure représentant l'arbre des balises. Seul cet ensemble d'ancêtres serait utilisé pour engendrer de nouvelles balises jusqu' à ce que la valeur de $EXPLORE(t)$ redevienne inférieure ou égale à la valeur de $EXPLORE(k-1)$, c'est-à-dire jusqu'à ce qu'on arrive à la résolution de $EXPLORE(k-1)$.

2. Une autre solution serait de considérer que la recherche doit se poursuivre à résolution constante et de ne prendre en compte dans la suite que les balises L_j avec $j < (k-1)$ pour lesquelles $EXPLORE(j) \geq EXPLORE(k)$

9.1.3 Etude sur le rebondissement

Un aspect important de cette thèse restant à approfondir est la technique du rebondissement. Le rebondissement est apparu expérimentalement comme une technique très efficace pour explorer la post-image et pour agrandir la pré-image. La question reste de savoir pourquoi ? Nous rappelerons d'abord la technique du rebondissement, puis nous exposerons quelques éléments de réponse basés sur les chaînes de Markov.

Etant donné une configuration q_{m-1} nous pouvons calculer l'intervalle $A_m = [x_i^{min}, x_i^{max}]$ pour le degré de liberté i. Il correspond au débattement maximal dans le sens négatif (x_i^{min}), et dans le sens positif (x_i^{max}). L'idée générale de la fonction *rebond* est de trouver la configuration q_m, atteinte en partant de la configuration q_{m-1} et en se déplaçant uniquement sur le degré de liberté i d'une distance x_i^j dans un intervalle A_m (voir figure 4.4 page 72). Si au moment de parcourir une distance $|\delta| < |x_i^j|$ on arrive à x_i^{max} ou x_i^{min}, la trajectoire "rebondit" et continue dans le sens inverse. On exécute alors cette même procédure pour le reste de la valeur x_i^j, c'est-à-dire pour $(x_i^j - \delta)$.

Le rebondissement augmente la probabilité d'explorer les différentes régions de la post-image. Par exemple, considérons l'espace des configurations discrétisé de la

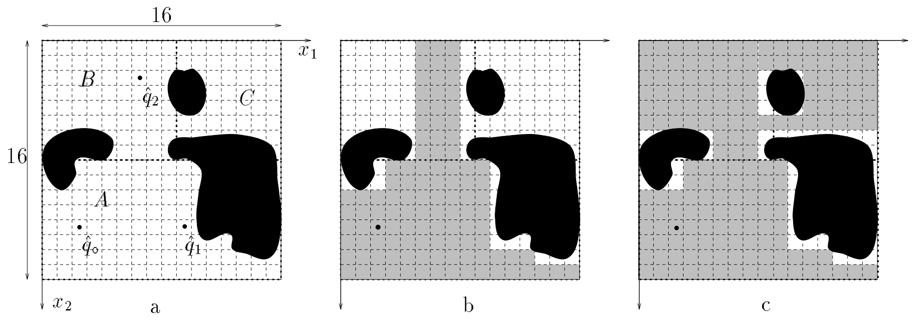


FIG. 9.2 - Espace dicrétisé et les chaînes de Markov.

figure 9.2a. Sur cet espace toute trajectoire Manhattan d'ordre 1 dans $\mathcal{C}_{libre}$ peut être codée par un point de $S = [0, 1, 2, \ldots 16] \times [0, 1, 2, \ldots 16]$.

Par exemple, l'extrémité de la trajectoire $(10, 10)$ est q_2 ou q_1 selon que l'on utilise ou non le rebondissement (voir figure 9.2a). Nous avons divisé cet espace en trois régions A, B et C. Il est facile de voir que la probabilité de passer de la région A à la région B est plus grande avec le rebondissement. En effet, la configuration q_2 peut être codée de deux façons différentes $(10, 10)$ et $(10, 4)$.

Les différentes régions de l'espace libre représentent les sommets (ou états) d'une chaîne de Markov. Dans notre exemple, nous pouvons considérer trois sommets, correspondant aux régions A, B et C. La probabilité de transition de A à B est alors égale à la proportion de l'espace S codant des trajectoires dont l'extrémité se trouve en B. Cette probabilité augmente lorsqu'on utilise le rebondissement.

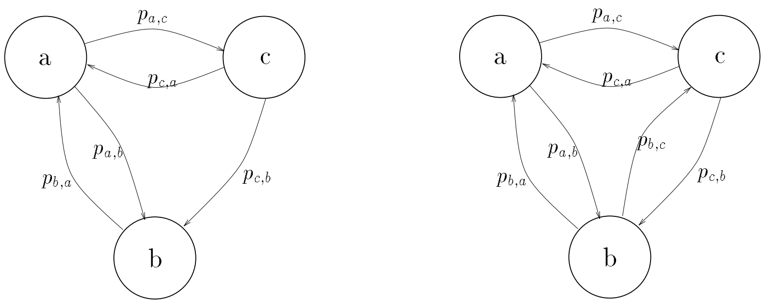
9.1.4 Etude sur l'ordre des trajectoires

Ce que nous venons de dire pour les rebondissements nous pouvons aussi le dire sur l'ordre[1] des chemins Manhattan utilisés.

La figure 9.2b montre la région de l'espace libre qui est accessible à partir de $\hat{q}_\circ$ et avec des chemins Manhattan d'ordre 1 et la figure 9.2c celle accessible avec des trajectoires Manhattan d'ordre 2.

A chaque région de l'espace libre correspond un sommet de la chaîne de Markov, les arcs sont définis en fonction de l'ordre k des trajectoires utilisées. Un arc existe d'un sommet i à un autre j si et seulement s'il est possible de passer de la région représentée par le sommet i à la région représentée par le sommet j avec une trajectoire Manhattan d'ordre k. La figure 9.3a montre la chaîne de Markov obtenue

[1]Rappel : l'ordre est le nombre de chemins Manhattan élémentaires composant à une trajectoire.



a b

FIG. 9.3 - Chaîne de Markov.

à partir de l'environnement de la figure 9.2a en utilisant des chemins Manhattan d'ordre 1 et la figure 9.3b en utilisant des chaînes d'ordre 2. La probabilité de transition $p_{i,j}$ est égale à la proportion de l'espace de recherche S codant une trajectoire du sommet i au sommet j. La valeur de chaque probabilité $p_{i,j}$ augmente lorsqu'on augmente l'ordre de la trajectoire.

9.1.5 Changement de technique d'optimisation

Un des principaux reproches fait à notre méthode est l'utilisation des algorithmes génétiques comme technique d'optimisation[34]. L'argument défendu est l'absence de résultats sur la convergence de cette méthode. Nous contestons cet argument à ceux qui ne l'appliquent pas aux méthodes basées sur l'utilisation du “ recuit simulé ” qui souffre en fait du même défaut.

1. La convergence du recuit simulé est démontrée sous des conditions bien précises de décroissance de la température qui ne sont pas appliquées en pratique car elles rendent l'algorithme inutilisable et peu différent d'une recherche aléatoire.

2. Il existe une preuve similaire pour les algorithmes génétiques mais elle a aussi peu d'intérêt pratique.

Nous pensons cependant qu'il est possible d'effectuer directement une descente de gradient pour optimiser $EXPLORE$ et $SEARCH$ car nous pensons qu'il est possible de calculer la dérivée de ces deux fonctions en tout point.

9.1.6 Utilisation d'un modèle hiérarchique

Un des problèmes majeur concernant l'utilisation de notre planificateur dans une application industrielle est le nombre de paires (articulation - obstacle) mises en jeu dans l'évaluation d'une trajectoire. Dans notre expérimentation, ce nombre ne dépassait pas la centaine alors qu'il peut atteindre plusieurs millions dans une application industrielle réelle. Une amélioration possible de notre système serait l'utilisation d'un modèle hiérarchique plus élaboré permettant une validation plus rapide des trajectoires[28].

9.1.7 Utilisation d'autres techniques de génération de trajectoires

Un point intéressant à explorer est l'utilisation d'autres techniques de génération de trajectoires permettant de valider facilement un déplacement local. Une possibilité prometteuse est l'utilisation d'approches comme celle proposée par Faverjon et Tournassoud [29]. Cette approche (la méthode des contraintes) permet d'engendrer dans un temps raisonnable des trajectoires pour des robots redondants placés dans des environnements complexes. Nous pouvons alors imaginer d'utiliser cette approche à la fois pour $SEARCH$ et $EXPLORE$.

9.2 Conclusions

9.2.1 Sur le plan théorique

Nous avons développée une technique pour la planification automatique en robotique : l'algorithme Fil d'Ariane. Cette technique est composée de deux sous-algorithmes, $EXPLORE$ et $SEARCH$. L'algorithme $EXPLORE$ permet d'approcher à ε-près tout espace S chemin connecté. Nous avons défini une approximation à ε-près comme un ensemble EL de points de S tel que, pour toute configuration q appartenant à S, il existe un point p de EL à une distance inférieure ou égale à ε. Nous avons appelé EL l'ensemble des balises. L'algorithme SEARCH est un algorithme de planification local qui exécute la recherche d'un chemin d'une configuration initiale à une configuration finale.

Les algorithmes $EXPLORE$ et $SEARCH$ ont été exprimés comme des problèmes d'optimisation sans contrainte sur $\mathbb{R}^{n*\ell}$ où n est la dimension de l'espace des commandes utilisées pour parcourir S et ℓ l'ordre des trajectoires que l'on désire utiliser. Grâce à une méthode de décodage : " le rebondissement ", il est possible de transformer tout vecteur $\hat{x} \in \mathbb{R}^{n*\ell}$ dans l'ensemble image d'un chemin $\hat{\gamma}$ de S. On dit alors que $\hat{x}$ code $\hat{\gamma}$.

Le but de l'algorithme Fil d'Ariane a été défini comme suit : à partir d'une configuration initiale $\hat{q}_\circ$ construire une approximation EL de l'espace de recherche S, c'est le rôle de la fonction $EXPLORE$. Le but de l'algorithme $SEARCH$ est de vérifier si la nouvelle balise placée par $EXPLORE$ appartient à la pré-image du but.

Nous avons montré que l'algorithme Fil d'Ariane est complet pour une résolution donnée [2]. Cette dernière propriété vient du fait que l'algorithme $EXPLORE$ assure que : s'il existe un chemin d'une configuration initiale $\hat{q}_\circ$ à une configuration finale $\hat{q}_\bullet$ et que ce chemin est inclus dans un "tube" de rayon ε appartenant entièrement à l'espace libre, alors il existe un temps finit T tel que $EXPLORE$ est capable de construire un chemin de $\hat{q}_\circ$ à $\hat{q}_\bullet$. L'algorithme $SEARCH$ est alors une amélioration de l'algorithme $EXPLORE$ qui permet d'accélérer la construction d'un chemin de $\hat{q}_\circ$ à une configuration particulière $\hat{q}_\bullet$.

9.2.2 Sur le plan expérimental

Nous avons montré la validité de l'Algorithme Fil d'Ariane en développant deux planificateurs. Le premier est un planificateur de trajectoires pour un robot mobile holonome. Le deuxième un planificateur de trajectoires pour un bras manipulateur à six degrés de liberté placé dans un environnement dynamique. L'algorithme a montré sa capacité à réagir à des changements rapides et imprévisibles de l'environnement, c'est-à-dire que l'algorithme peut-être utilisé comme un planificateur réactif pour certains problèmes. Lorsqu'une trajectoire est calculée et qu'un objet change de position, il est possible de re-planifier, avec un temps de réaction faible, une nouvelle trajectoire.

Nous avons utilisé les algorithmes génétiques comme technique d'optimisation, d'une part parce qu'ils s'adaptent très bien aux architectures parallèles et d'autre parce qu'ils sont appropriés au problème d'optimisation posé.

Dans le cas du bras manipulateur une version parallèle du système a été implantée. Trois niveaux de parallélisme ont été établis. Le premier niveau est chargé de l'exécution des algorithmes $SEARCH$ et $EXPLORE$. Le deuxième niveau exécute les algorithmes génétiques chargés de l'optimisation. Finalement, le troisième niveau contient la fonction d'évaluation. Cette fonction est basée sur le calcul de débattements entre parallélépipèdes.

[2] "resolution complet"

9.2.3 Discussion

Dans cette thèse nous nous sommes intéressés au problème de la planification de trajectoire en robotique. Dans le cadre de cette étude nous avons défini un planificateur comme un programme capable de fournir une liste de commandes de déplacement réalisables au sens de notre modèle et permettant d'atteindre un certain objectif. D'autres problèmes de planification trouvés en robotique rentrent facilement dans ce cadre : c'est le cas par exemple de la planification de la saisie d'objet, de la synthèse des mouvements fins d'assemblage ou de la manipulation en général.

L'hypothèse centrale de ce document est de montrer que le problème de la planification peut, par un changement d'espace approprié, se ramener à la résolution d'un problème d'optimisation et que l'on peut facilement engendrer un planificateur dès lors que l'on sait prévoir la faisabilité d'une commande dans l'espace ou l'on définit les buts. Nous avons vérifié cette hypothèse pour la planification de trajectoire et nous pensons qu'elle peut s'appliquer aux problèmes de robotique évoqués plus haut pour les raisons suivantes:

Admettons, pour un instant, que l'on puisse représenter tout chemin d'un espace comme un point dans un espace de Hilbert P de dimension fini, autrement dit que l'on puisse définir tout chemin comme la combinaison linéaire d'un ensemble fini de fonctions de [0,1] dans C. Alors, un point de cet espace de Hilbert représente un chemin dans cet espace, les chemins illégaux (passant par des *$\mathcal{C}$-obstacles*) définissent aussi des régions interdites : les *P-obstacles*. Sur cet espace, on peut définir une fonction de coût F qui associe à chaque point p une valeur correspondant à $+\infty$ si ce point appartient à un P_obstacle et à la distance de l'extrémité du chemin au but sinon. Il devient clair que le problème de la recherche d'un plan est un problème de minimisation sur P. Il nous suffit, en effet, de trouver un point p qui annule cette fonction pour avoir une solution à notre problème de planification.

Nous pouvons maintenant faire les observations suivantes:

1. On peut obtenir la valeur de F en tout point p de H. Pour cela il nous suffit d'utiliser notre modèle et d'appliquer le plan correspondant à p à partir de la position initiale. Cette importante caractéristique nous permet d'attaquer le problème de la planification dans des espaces de grande dimension.

2. Il existe une technique simple permettant de faire “fondre” les P_obstacles et de rendre la fonction F continue presque partout : **les rebondissements**. Cette technique consiste à faire rebondir les trajectoires non valides sur les *$\mathcal{C}$-obstacles*. Elle revient à appliquer une fonction R de H dans H qui transforme tout point appartenant à un P_obstacle en un point de P_{libre} (le complémentaire des P_obstacles dans H). La fonction R est facilement calculable et le

remplacement de F par R o F facilite grandement la recherche d'un optimum.

3. R o F s'annule sur des nappes de H. Cette propriété traduit la possibilité de passer continûment sur une infinité de chemins amenant au même but. Ces nappes peuvent être distinctes et reparties dans H (notion de classe d'homotopie). La surface et le nombre de ces nappes reflètent la difficulté du problème. En pratique, ces nappes sont "éparpillées" dans H et de grandes dimensions. Elles peuvent être découvertes par des techniques d'optimisation non exactes.

La méthode de planification que nous avons proposée, réunit les avantages suivants :

1. elle ne fait pas appel au calcul de l'espace des configurations, mais elle peut en fournir une approximation,
2. elle s'adapte naturellement à la difficulté du problème en trouvant la "bonne" approximation de l'espace des configurations,
3. elle est facile à implémenter même sur des machines massivement parallèles,
4. et elle peut être utilisée dans d'autres problèmes de planification que celui de la planification de trajectoires.

Les remarques précédentes montrent différents axes de recherche permettant de l'améliorer. En conclusion nous voyons l'algorithme Fil d'Ariane comme une technique générique de planification.

Annexes

Annexe A

Synthèse des travaux existant en planification de trajectoires

Dans cette annexe nous allons situer notre travail en nous intéressant tour à tour : aux problèmes posés, aux résultats escomptés et aux méthodes employées en planification de trajectoires. Pour cela nous utiliserons la classification proposée par Hwang et Ahuja dans [36].

A.0.4 Classification des problèmes par type d'environnement

On peut classifier le problème de la planification de trajectoires selon les caractéristiques de l'environnement :

- **statique**, toute l'information concernant les obstacles est connue a priori et les trajectoires sont élaborées à partir de cette information,
- **dynamique**, seule une partie de l'information sur les obstacles est connue comme par exemple leurs vitesses et positions initiales

- **avec des objets déplaçables**, les robots peuvent changer la position des certains objets de l'environnement,

- **avec contraintes** [1], se réfère aux problèmes où il faut prendre en compte les contraintes mécaniques du robot, vitesse, accélération, courbure des trajectoires etc..., et les contraintes fixées par l'utilisateur.

Problèmes de planification avec un environnement statique

Ce type de problème consiste à planifier la trajectoire d'un robot qui évolue dans un environnement où l'on connaît a priori la position des obstacles. Un modèle statique de l'environnement est utilisé pour planifier les trajectoires du robot. Les algorithmes utilisés pour résoudre ce type de problème sont très souvent utilisés pour construire des planificateurs plus généraux avec des environnements dynamiques ou avec des objets déplaçables. Ils peuvent aussi être utilisés comme une première phase de résolution pour les problèmes avec contraintes. C'est à ce type de problème que nous nous attaquons dans ce travail.

Problèmes de planification avec environnement dynamique

Dans les problèmes d'environnement dynamique, nous trouvons deux situations différentes : la première si la position des obstacles au temps t est connue ou prévisible, la deuxième si cette information est inconnue. La première situation se résume, en termes généraux, à résoudre un problème de type environnement statique. Dans ce cas, on peut augmenter la dimension du problème en intégrant celle du temps [30]. La seconde situation est plus difficile car, si les changements de l'environnement sont inconnus, nous sommes obligés de calculer une trajectoire à partir de l'information disponible. Ainsi, à mesure que le robot se déplace sur la trajectoire précédemment calculée, il peut découvrir de nouvelles caractéristiques de l'environnement. Une vérification de la validité de la trajectoire est alors effectuée continuellement. Si la trajectoire précédemment calculée n'est plus valide, il faut en replanifier une nouvelle à partir de la position courante. Les nouvelles trajectoires sont planifiées tout en considérant une mise à jour de l'environnement. En résolvant rapidement une série de problèmes statiques on peut, comme c'est le cas pour notre méthode, répondre à certains problèmes mettant en jeu des environnements dynamiques.

[1] Dans la littérature anglaise cette classification correspond à : *stationary, time-varying, movable-object* et *constrained* respectivement.

Problèmes de planification avec des objets déplaçables

Lorsque le robot peut déplacer des objets, on dit que l'environnement contient des objets déplaçables. Une formulation théorique de ce problème est présentée en [8].

Planification de trajectoires avec contraintes

Parmi les contraintes les plus répandues en planification de trajectoires on trouve : les contraintes de non-holonomie et les contraintes dues aux limitations de vitesse ou d'accélération des moteurs du robot. Un système est dit non-holonome lorsque les contraintes cinématiques du robot sont non-intégrables [42][43][78]. Des contraintes de non-holonomie sont associées, par exemple, à une voiture dont la structure mécanique et le rayon de braquage imposent un mouvement tangent à sa direction en un point donné; c'est-à-dire que toute trajectoire sans collision dans l'espace des configurations admissible ne correspond pas nécessairement à une trajectoire réalisable.

A.0.5 Classification des problèmes par type de robot

Le problème de la planification de trajectoires peut aussi être classifié par les types de robot considérés.

1. **robots rigides**, dans ce problème on considère un objet rigide pouvant se déplacer sans contraintes cinématiques dans un espace 3D ou 2D. Ce type de problème est aussi connu comme *le problème du déménageur de piano* [74]. Il consiste à planifier des trajectoires pour un objet non déformable dans l'espace ambiant.

2. **robot manipulateur**, dans ce problème on considère maintenant un ensemble de corps articulés. Les articulations sont soit des rotations soit des translations. Elles permettent au robot de changer sa topologie pour exécuter différentes tâches. Chacune des articulations indépendantes définit un degré de liberté du robot. L'essentiel de notre travail porte sur la réalisation d'un planificateur pour ce type de robot.

3. **Robot mobile**, ils sont essentiellement considérés comme des solides pouvant se déplacer sur une surface 2D. Ils peuvent être de forme quelconque ou assimilé à un cercle si l'on décide ou non de tenir compte de l'orientation. Nous avons expérimenté notre méthode sur des robots mobiles de forme rectangulaire sans contrainte cinématique.

A.1 Classification des approches de la planification

On peut classifier les approches de la planification de trajectoires par le type de résultats obtenus et par le type de méthodes employées :

A.1.1 Classification par les résultats

On distingue quatre types de résultat possibles pour une méthode de planification de trajectoires :

1. **Méthodes exactes** : Une approche exacte garantit la résolution du problème s'il existe une solution. Si il n'en existe pas, alors l'approche démontre l'inexistence de celle-ci.
2. **Méthodes heuristiques** : Une approche heuristique se base sur certaines hypothèses implicites faites sur le problème[63]. Si ces hypothèses ne sont pas vérifiées pour un problème donné alors la méthode peut tout simplement échouer bien qu'une solution existe.
3. **Méthodes complètes à une résolution donnée** : La complétude à une résolution donnée est reliée à la division de l'espace de recherche. Une méthode est complète à résolution donnée si elle garantit de trouver toutes les solutions qui auraient été trouvées par une recherche exhaustive dans l'espace de recherche discrétisée. L'algorithme File d'Ariane fait parti de ce type de méthode.
4. **Méthodes ayant une complétude probabiliste** : Une approche a une complétude probabiliste si à mesure que le temps passe la probabilité de trouver une solution tend vers 1.

A.1.2 Classification par type d'approche

Les approches en planification de trajectoires peuvent être classées en trois groupes :

- **les approches globales** prenant en compte toute l'information sur l'environnement.
- **les approches locales** capables de planifier des trajectoires au voisinage de la position courante du robot.
- **les méthodes mixtes** combinant les deux approches précédentes.

Avant d'introduire ces différentes approches nous rappelons brièvement la notion d'espace des configurations.

La *configuration* d'un objet (ou robot) ayant une forme donnée est un ensemble de paramètres indépendants qui caractérisent la position de chacun des points de l'objet. Par exemple pour un objet en 2D il nous faut trois paramètres, la position en x la position en y et l'orientation. Pour un bras manipulateur planaire à deux composants il nous faut deux valeurs angulaires, l'angle de la première articulation et l'angle de la deuxième articulation. De cette façon, la position d'un robot dans un environnement de travail est représentée par un point de $\mathbb{R}^n$. On appelle n le nombre de *degrés de liberté* du robot, c'est-à-dire que n est le nombre de paramètres minimum qui spécifient complètement la position de tous les points du robot.

L'ensemble de toutes les configurations est appelé *l'espace des configurations* noté $\mathcal{C}$. Dans cet espace, on définit *l'espace libre* comme l'ensemble des configurations de $\mathcal{C}$ qui ne provoquent pas une collision dans l'espace de travail du robot. On appelle $\mathcal{C}$-*obstacles* l'ensemble des configurations provoquant une collision. On peut aussi dire que les $\mathcal{C}$-*obstacles* est le complémentaire dans $\mathcal{C}$ de l'espace libre. Ainsi, un chemin libre de collision exécuté par le robot dans son environnement peut être transformé en une courbe de l'espace des configurations. En général l'inverse n'est pas toujours vrai, ceci est du aux contraintes cinématiques du robot, c'est-à-dire qu'il n'est pas toujours possible de transformer une courbe de *l'espace libre* en une suite de mouvements valides du robot dans l'espace de travail.

Les approches globales

Un approche globale construit une représentation de l'espace des configurations, à partir du modèle complet de l'environnement. En générale ce type d'approche transforme le problème de planification en un problème de recherche dans un graphe. Les approches globales sont regroupés classiquement en deux catégories :

1. **Squelette ou de rétraction.** Cet approche cherche à construire une représentation de l'espace des configurations avec des sous-espaces de dimension plus petite que celle de l'espace des configurations. Les propriétés de connexion sont gardées dans une nouvelle représentation appelée *squelette ou rétraction*. Ainsi, si deux configurations appartiennent à la même composante connexe dans l'espace des configurations elles appartiennent aussi à la même composante connexe dans la rétraction.

2. **Décomposition cellulaire.** Cette approche consiste à diviser l'espace de recherche en sous-espaces appelés cellules. La planification d'une trajectoire est alors la construction d'un chemin entre deux sous-espaces. Chaque cellule

libre d'obstacles représente un sommet. Deux sommets sont connectés entre eux si les cellules représentant les sommets sont adjacentes.

L'approche de squelette ou de rétraction Dans cette approche, une représentation explicite des espaces libres connexes est utilisée. L'ensemble de trajectoires dans l'espace libre est rétracté ou réduit dans un graphe. Ce graphe représente toutes les régions de l'espace libre. De cette manière, toute trajectoire dans l'espace libre est représentée par un ensemble d'arcs et sommets. Pour chercher une trajectoire d'une configuration initiale à une autre finale, il suffit de projeter la configuration initiale et finale dans deux sommets du graphe. Après, une méthode de recherche est utilisée pour parcourir le graphe et trouver l'ensemble de sommets et arcs (représentant de sous-espaces de l'espace libre) permettant de construire la trajectoire.

Les approches existantes pour la construction d'une rétraction de l'espace libre sont :

- **Le graphe de visibilité**, utilisé pour des environnements composés d'obstacles polygonaux. Le graphe de rétraction est construit avec les sommets des objets, la configuration initiale et la configuration finale.
- **Les diagrammes de Voronoi**, consistant à construire une rétraction de l'espace libre à partir des point équidistants entre les obstacles.
- **La Méthode d'Avnaim et Boissonnat**, consistant à construir un graphe à partir du calcul de la frontière de l'espace libre.

Un des inconvénients des deux premières approches est qu'elles s'appliquent seulement à des espaces à deux dimensions. On peut trouver des études concernant le diagramme de Voronoi, le graphe de visibilité et la division convexe d'un espace en deux dimensions en [70]. L'inconvénient de la troisième est de ne pas s'appliquer en pratique à des problème considérant des robots ayant plus de trois degré de liberté. Nous présentons brièvement ces trois approches.

Le graphe de visibilité. L'idée principale de cette approche est créer un graphe avec les sommets des obstacles polygonaux, la position initiale et la position finale. Deux sommets sont connectés par une ligne droite si elle n'intercepte pas l'intérieur des obstacles. Ainsi, il faut trouver le chemin entre les sommets du graphe qui connecte le sommet initial au sommet final [40],[79],[76]. Des techniques de réduction du nombre de sommets du graphe sont généralement utilisées pour simplifier les recherches.

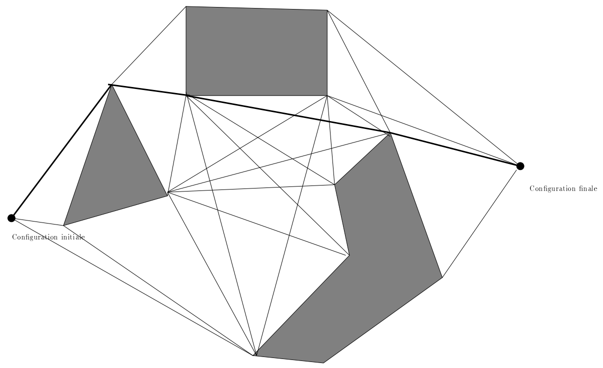


FIG. A.1 - Graphe de visibilité.

Un exemple de construction du graphe de visibilité ayant pour sommet les sommets des obstacles polygonaux et les positions initiale et finale est montré dans la figure A.1. Une des procédures de recherche les plus utilisée pour naviguer dans le graphe est l'algorithme $A^{\star}$[86]. Cet algorithme est une spécialisation de l'algorithme meilleur-d'abord qui permet de trouver le chemin le plus court dans un graphe d'un sommet initial à un autre final. En plus, il garantit de trouver le chemin s'il existe.

Les diagrammes de Voronoi. Il s'agit de l'utilisation des espaces libres obtenus par les diagrammes de Voronoi[70]. Dans un espace de travail encombré d'obstacles, le diagramme de Voronoi est un réseau de segments linéaires et paraboliques obtenus à partir des points équidistants entre les obstacles [79]. Cette approche permet de trouver des trajectoires éloignées des obstacles. La figure A.2 montre le diagramme de Voronoi obtenu dans un environnement polyédrique.

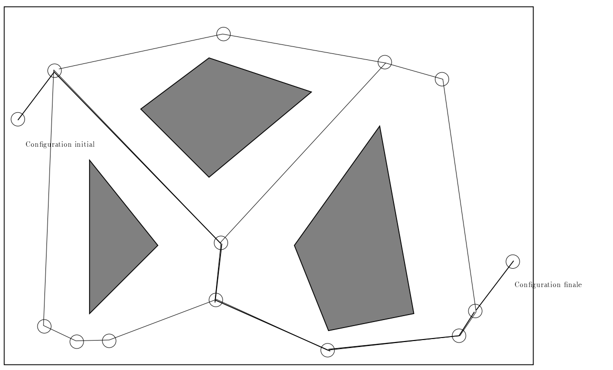


FIG. A.2 - Graphe de Voronoi.

La méthode d'Avnaim et Boissonat. Cette méthode est appliquée aux robots et objets polygonaux. Elle consiste à faire une décomposition de l'espace des configurations à partir des frontières de l'espace libre. Pour obtenir ces frontières, un étude systématique de tous les contacts entre le polygone constituant le robot et les obstacles est faite.

La décomposition cellulaire. La décomposition cellulaire consiste à diviser l'espace des configurations en un ensemble de sous-espaces et à trouver les relations d'adjacence entre eux [49],[75],[21]. La planification d'une trajectoire est alors la construction d'un chemin entre deux sous-espaces. Chaque cellule libre d'obstacles représente un sommet. Deux sommets sont connectés entre eux si les cellules représentant les sommets sont adjacentes. Les approches de décomposition en cellules sont classifiés en *exactes* et *approximatives*.

1. **Décomposition exacte :** Pour les approches exactes il s'agit d'obtenir une représentation exacte de toutes les cellules complètement libres d'obstacles (voir figure A.3).

2. **Décomposition approximative :** Dans ce cas on se contente de connaître un sous espace de l'espace libre que l'on peut par exemple représenté sous la forme de quadtree. Pour ce type de représentation on considère trois types de cellule : les *cellules libres*, les *cellules $\mathcal{C}$-obstacles* et les *cellules mixtes*. Les cellules libres sont des sous-espaces de l'espace libre, les cellules *$\mathcal{C}$-obstacles* sont de sous-espaces des *$\mathcal{C}$-obstacles* et les cellules mixtes contiennent un sous-espace de l'espace libre et un sous-espace des *$\mathcal{C}$-obstacles*. La figure A.4 montre ces trois types de cellules.

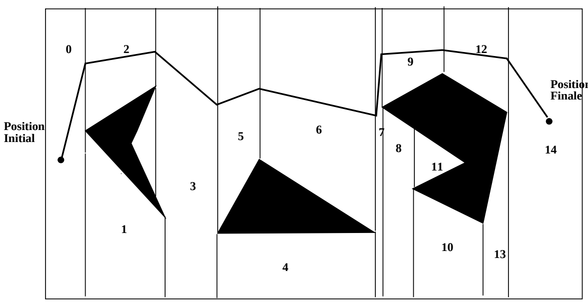


FIG. A.3 - Décomposition exacte.

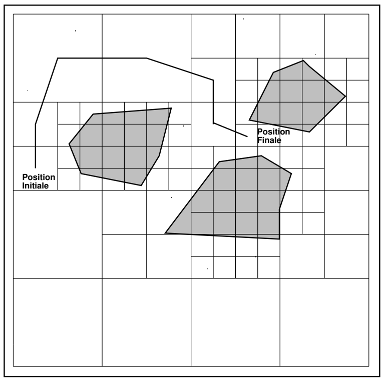


FIG. A.4 - Décomposition approximative.

Les approches locales

Ce type d'approche utilise une représentation locale de l'environnement pour essayer d'engendrer une trajectoire vers le but. Ces approches sont basées sur l'optimisation d'une fonctionnelle qui amène le robot de plus en plus près du but. En générale, elles sont utilisées lorsque le point de départ et le but sont des configurations voisines. Les approches locales sont aussi utilisées en combinaison avec les méthodes globales pour atteindre de sous-buts intermédiaires qui permettent l'amélioration de la recherche. Nous trouvons principalement deux approches locales :*les champs de potentiels* et *la méthode des contraintes*.

L'approche des potentiels. L'idée de cette approche est de considérer le robot comme une particule sous l'influence des forces d'attraction et répulsion. Ces forces sont engendrées respectivement par la position finale et les obstacles. La somme de ces deux forces peut être représentée par un champ de courbes équipotentielles [38] [39][40][66]. La particule (le robot) est muni alors d'une "énergie potentielle" en se trouvant dans un point "plus haut" que la position finale. "L'énergie potentielle" obligera la particule à descendre par un chemin libre de collision car la force de répulsion est infinie en chaque point qui forme un obstacle (voir figure A.5). La fonction potentielle prend un minimum local correspondant au but. De cette manière, la trajectoire atteindra le but. En comparaison avec d'autres approches, cette technique est très efficace, par contre, il est difficile de définir une fonction potentiel n'ayant qu'un minimum et par conséquent il est possible de tomber dans des minima locaux différents du but. L'approche peut être améliorée en utilisant des heuristiques pour éviter ce problème.

La méthode des contraintes. Ce type d'approche, proposée par Faverjon et Tournassound [29][85], représente localement l'espace des configurations par une liste de contraintes. Le problème de la planification de trajectoires est alors ramené à un problème d'optimisation sous contraintes où l'on cherche à minimiser une fonction de tâche. Cette fonction est définie à partir de la spécification du but à atteindre (but articulaire, cartésien etc...). La minimisation sous contraintes permet alors de déterminer localement la direction du mouvement que le robot doit effectuer pour atteindre le but sans toucher les obstacles. Cette méthode souffre aussi de la présence de nombreux minima locaux.

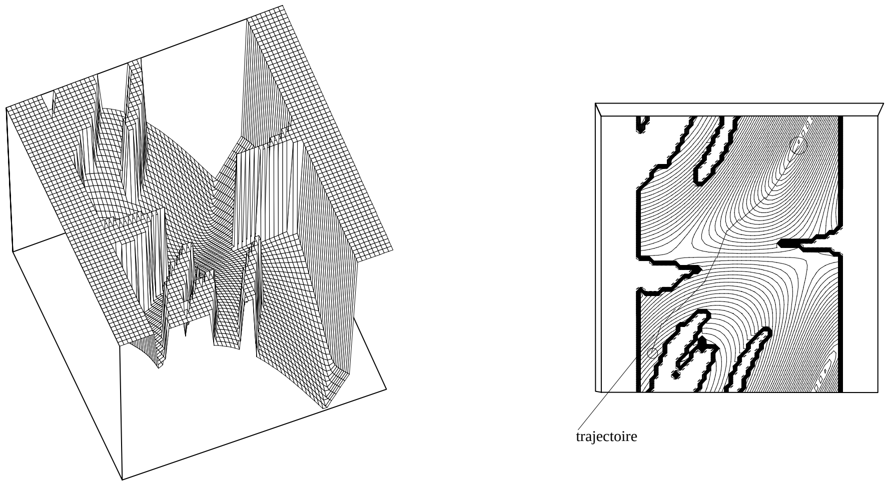


FIG. A.5 - Exemple de l'approche potentielle.

A.1.3 Complexité du problème

Dans ce paragraphe nous abordons le problème de la complexité de la planification de trajectoires[14]. Cette présentation est faite d'après la thèse de M. Pasquier [62]. Premièrement nous donnons les notions de base de complexité algorithmique, puis nous analysons la complexité du problème de la planification de trajectoires.

Lorsqu'on veut résoudre un problème avec un ordinateur il faut non seulement trouver un algorithme qui permet de trouver une solution, il faut aussi tenir compte des contraintes d'espace mémoire et de temps calcul. Ainsi, un algorithme doit être capable de trouver une solution qui soit calculable avec une place mémoire et un temps raisonnables.

Le temps d'exécution et la place mémoire utilisés par un algorithme déterminent la *complexité algorithmique*. La complexité de la résolution d'un problème est ainsi définie comme le nombre d'opérations requises (exprimées en fonction de la taille m des donnés) pour le résoudre.

La complexité d'un problème sera dite $O(f(m))$ si son temps d'exécution (ou espace mémoire) à le même comportement asymptotique que la fonction $f(m)$. Les algorithmes *polynomiaux* sont des algorithmes où la complexité peut être évaluée à un polynôme de degré constant connu, où le degré du polynôme est indépendant de m. Un algorithme non polynomiale est dit de complexité *exponentielle*.

En informatique on peut distinguer deux types de machine algorithmique :

– **les machines déterministes** qui peuvent être ramenées à un automate d'états qui à un moment donné, se trouve dans un seul état bien déterminé et dont l'action dépend que de celui-ci,

- **les non-déterministes** qui définissent aussi un automate, mais qui sont munis en plus d'une fonction dite de choix (aléatoire), et dont les actions ne dépendent donc pas uniquement de l'état dans lequel il se trouve mais également du hasard.

On distingue alors une hiérarchie de trois classes de problème allant des plus simples aux plus difficiles:

- **La classe P** comprend les problèmes pour lesquels on connaît un algorithme dont la résolution demande un temps polynomial sur une machine déterministe.
- **La classe NP** sont les problèmes pouvant être résolus en temps polynomial sur une machine non déterministe.
- **La classe PEspace** regroupe les problèmes qui peuvent être résolus avec une mémoire polynomial sur une machine non déterministe.

La complexité algorithmique de la planification de trajectoire à surtout été étudiée pour le problème du déménageur de piano. Il a été prouvé par Schwartz et Sharir [74] que le problème de planification de trajectoires d'un robot à n degrés de liberté est calculable, c'est-à-dire qu'il est déterministe-polynomial ou de classe P.

La méthode utilisée pour arriver à ce résultat est la décomposition cylindrique d'ensembles semi-algébriques de Collins. De cette façon, l'évaluation de la complexité, pour un manipulateur à n degrés de liberté dans un univers défini par m faces, est $m^{(2*n+6)}$. Cependant, ce résultat n'est, hélas, d'aucune utilité pratique étant difficile à implanter de manière efficace. Ce résultat peut néanmoins être utilisé comme borne supérieure au problème de planification de trajectoires.

Une amélioration à la borne donnée par Schwartz et Sharir a été donnée par Canny [14]. Dans son travail, il montre que le problème général pour un nombre de degrés de liberté non borné est de classe $PEspace$. Il décrit une méthode générale qui donne un algorithme simplement exponentiel. D'autres résultats ont également été obtenus pour d'autre types de problèmes. Ainsi, beaucoup de problèmes de planification de trajectoire ont été prouvés ayant une complexité de classe NP ou $PEspace$.

Annexe B

Calculs Géométriques

L'opération élémentaire autour de laquelle est organisée le calcul du débattement d'un degré de liberté du robot est le calcul du débattement entre deux parallélépipèdes: l'un mobile PM et l'autre fixe PO. Deux principes ont été retenus pour accélérer la vitesse de calcul de ce type de débattement:

1. **Le changement de coordonnées paresseux:** le calcul du débattement proprement dit ne peut débuter qu'après avoir exprimé les paramètres de PO dans le repère de l'articulation. En général il n'est pas nécessaire de faire cette transformation pour tous les paramètres avant de décider que les deux parallélépipèdes ne peuvent rentrer en collision.

2. **L'utilisation de filtres:** Des tests, économiques en temps de calcul, permettent de filtrer de façon quasi instantanée l'absence d'interaction possible entre PM et PO.

B.1 Entités Géométriques

Les parallélépidèdes sont représentés par leur frontière (BREP). Trois types d'entités géométriques sont utilisées pour les modéliser:

1. Les points pour modéliser les sommets.
2. Les droites pour modéliser les arêtes.
3. Les plans pour modéliser les faces.

Ces entités géométriques sont représentées dans l'espace par leurs coordonnées de Plücker à savoir:

- **Point:** $M = (x, y, z)$.
- **Droite:** $L = (u, v)$ ou u est le vecteur unitaire directeur de la droite et v un vecteur tel que si M est un point de la droite $M \wedge u = v$ ($M \wedge u$ représente le produit vectoriel entre M et u).
- **Plan:** $P = (u, a)$ ou u est le vecteur unitaire normal au plan et ou a est un scalaire tel que si M est un point du plan $M \bullet u = a$ ($M \bullet u$ représente le produit scalaire entre M et u).

Les formules de changement de coordonnées pour une translation t d'un repère R_1 vers un repère R_2 sont données par:

- **Point :** $m_2 = m_1 - t_1$ où m_2, m_1, t_1 représentent respectivement les coordonnées du point M dans les repères R_2, R_1 et les coordonnées du vecteur t dans le repère R_1.
- **Droite :** $L_2 = (u_1, v_1 - t_1 \wedge u_1)$.
- **Plan :** $P_2 = (u_1, a - u_1 \bullet t_1)$.

Les formules de changement de coordonnées pour une rotation H d'un repère R_1 vers un repère R_2 sont données par:

- **Point :** $m_2 = H(m_1)$.
- **Droite :** $L_2 = (H(u_1), H(v_1))$.
- **Plan :** $P_2 = (H(u_1), a)$.

B.2 Calculs de débattement

Dans ce paragraphe nous nous limitons au calcul de débattement entre deux parallélépipèdes, dans notre application ce calcul doit être répété pour toutes les paires: parallélépipède mobile - parallélépipède obstacle. Ces calculs peuvent être menés en parallèle.
Pour déterminer le débattement, trois types de contact doivent être envisagés:

1. **Type A :** Un sommet de l'objet mobile contre une face de l'objet fixe.
2. **Type B :** Un sommet de l'objet fixe contre une face de l'objet mobile.
3. **Type C :** Un arête de l'objet mobile contre une arête de l'objet fixe.

Étant donné l'aspect symétrique d'un débattement il nous suffit de considérer les contacts de type A et les contacts de type C, les calculs sur les contacts de type B se déduisant de ceux de type A. Une première série de calculs concerne les "supports" infinis des entités géométriques face et arête. Nous cherchons la rotation d'angle θ autour du vecteur Z de l'articulation qui réalise ces types de contact

- **Type A :** Le plan (u,a) de l'articulation contient le point m du mobile.

$$H(\theta, m) \bullet u = a$$

$$(m_x u_x + m_y u_y)\cos(\theta) + (u_y m_x - u_x b_y)\sin(\theta) = a - u_z m_z$$

- **Type B :** La droite(u,v) de l'articulation intersecte la droite (s,w) du mobile.

$$H(\theta, w) \bullet u + H(\theta, s) \bullet v = 0$$

$$(u_x w_x + a_y w_y + v_x s_x + v_y b_y)\cos(\theta) + (u_y s_x - u_x s_y + v_y s_x - v_x s_y)\sin(\theta) + v_z s_z + u_z w_z = 0$$

La résolution de ces deux équations sur toutes les paires d'entités géométriques nous permet de calculer les débattements entre les supports "infini" de ces entités.
La résolution de l'équation

$$a\cos(\theta) + b\sin(\theta) = c$$

se fait en posant $d = \sqrt{(a^2 + b^2)}$, $c\prime = c/d$ et $\phi = \arctan(b/d, a/d)$. On obtient une nouvelle équation:

$$\cos(\phi)\cos(\theta) + \sin(\phi)\sin(\theta) = c\prime$$

qui admet deux solutions

$$\theta = \phi \pm \arccos(c\prime)$$

Il faut maintenant vérifier que le débattement correspond bien à un contact sur les frontières du parallélépipède. On obtient les points de contact par:

- **Type A :** $p = H(-\theta, m)$.
- **Type C :** on pose $(sp, wp) = (H(-\theta, s), H(-\theta, w))$ et $w_1 = wp - u \wedge v \wedge sp$ On obtient
$$p = \frac{w_1 \bullet w_1}{u \bullet (sp \wedge w_1)} u + u \wedge v$$

Pour vérifier leur appartenance à une face ou à une arête on utilise les formules suivantes:

- **appartenance à une face ayant** $(s1, s2, s3, s4)$ **comme sommets:**
$$p \wedge s1 \bullet z > 0 \ et \ p \wedge s2 \bullet z > 0 \ et \ p \wedge s3 \bullet z > 0 \ et \ p \wedge s4 \bullet z > 0$$
- **appartenance à arête ayant** $(s1, s2)$ **comme sommets:**
$$s1 - p \bullet s2 - p < 0$$

Les figures 7.8 7.9 7.10 représentent les débattements calculés pour deux parallélépipèdes dans les cas A, B et C (Voir pages 116,117 et 118).

Bibliographie

[1] A.V. Aho, J.E. Hopcroft, J.D. Ulman : *The design and analysis of computer algorithms*, Reading, MA: Addison-Wesley, 1974.

[2] Juan Manuel Ahuactzin, *Utilisation d'algorithmes génétiques pour la planification de trajectoires en robotique.* Rapport de D.E.A. Lifia, Grenoble France, juin 1991.

[3] Juan Manuel Ahuactzin, El-Gazali Talbi, Pierre Bessiere, Emmanuel Mazer. *Using Genetic Algorithms for Robot Motion Planning*, Journées "Raisonnement Géometrique : de la perception vers l'action", Grenoble, France septembre 1991.

[4] J.M. Ahuactzin, E. Mazer, P. Bessiere, E.G. Talbi. *Using Genetic Algorithms for Robot Motion Planning*, European Conference on Artificial Intelligence, Vienna, Austria, aout 1992.

[5] Juan Manuel Ahuactzin, El-Gazali Talbi, *Un algorithme génétique pour la planification stochastique de trajectoires en robotique.* Premiéres rencontres nationales des jeunes chercheurs en intelligence artificielle. Rennes, France, septembre 1992.

[6] Juan Manuel Ahuactzin, E. Mazer, P. Bessiere, E.G. Talbi. *Using Genetic Algorithms for Robot Motion Planning,* en,*Lecture Notes in Computer Science 708, Geometric Reasoning for perseption an Action, pp.84-93* Springer-Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest, 1993.

[7] Juan Manuel Ahuactzin, El-Gazali Talbi, Thierry Chatroux, Pierre Bessiere, Emmanuel Mazer. *A Massively Parallel Implementation of the Ariadnés Clew Algorithm*, Intelligent Robotic System9́3, Zakopane, Poland, 20-24 juillet 1993.

[8] Alami R., Siméon, T., Laumond, J.P. : A Geometric Approach to Planning Manipulateur Tasks. The Case of discrete placements and graps, Proceedings of the 5th International Symposium of Robotic Research, Tokyo, août 28-31, MIT Press, Cambridge, Mass.

[9] E. J. Anderson, M.C. Ferris : *A genetic algorithm for the assembly line balancing problem*, Tech. Rep. No. 926, University of Wisconsing-Madison, mars 1990.

[10] Jérôme Barraquand, Jean-Claude Latombe: *Robot Motion Planning A Distributed Representation Approach*, Robotics Laboratory, Computer Science Departement, Stanford University.

[11] Jérôme Barraquand and Jean-Claude Latombe: *A Monte-Carlo Algorithm for Path Planning With Many Degrees of Freedom*,Proceedings of the 1990 IEEE International Conference on Robotic and Automation, Ohio, USA - mai 1990.

[12] Pierre Bessière, Juan Manuel Ahuactzin, El-Ghazali Talbi, Emmanuel Mazer. *The "Ariadnés clew algorithm": Global Planning with local methods"*, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems'93 (IROS'93), Yokohama, Japan juillet 26-30, 1993.

[13] Paul Borrel. *Contribution à la modelisation geometrique des robots manipulateurs. Appliation a la conception assistée par ordinateur*, Thèse d'Etat, USTL, Montpellier, France juillet 1986.

[14] John Francis Canny. *The complexity of robot motion planning*, PhD Thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, mai 1987

[15] John F. Canny: *An opportunistic Global Path Planner*, Proceedings of the 1990 IEEE International Conference on Robotic and Automation, Cincinnati, Ohio, USA - mai 1990.

[16] Daniel J. Challou, Maria Gini and Vipin Kummar. *Parallel Search Algorithms for Robot Motion Planning*, Proceedings of the 1993 IEEE International Conference on Robotic and Automation, Atlanta, Georgia, USA - mai 1993.

[17] Thierry Chatroux, *Algorithmes génétiques parallèles pour la planification de trajectoires de robots en environnement dynamique*, Mémoire d'ingénieur C.N.A.M., Grenoble, France 1993.

[18] C. Chen, Yong K. Hwang: *SANDROS: A Motion Planner with Performance Proportional to Task Difficulty*,Proceedings of the 1992 IEEE International Conference on Robotic and Automation, Nice, France - mai 1992.

[19] Claude Chiru : *Application des algorithmes génétiques parallèles au problème de la cinematique inverse en robotique* Travail de diplôme d'ingénieur en informatique, Ecole polytechnique fédérale de Lausanne. Travail réalisé au LGI/IMAG Grenoble, France février 1994.

[20] Y. Davidor: *Analogous Crossover*, Third International Conference on Genetic Algorithms, 1989.

[21] Subbarao Kambhampati, Larry S. Davis: *multiresolution path planning for mobile robots*, IEEE Robotic and Automation juin 1986.

[22] F. De la Rosa and C. Laugier and J. Nájera : *Exploring the Contact Space to Plan Robot Motions under Geometric Uncertainty Constraints*,International Workshop on Intelligent Robotic Systems'93, Zakopane, Poland, juin 1993.

[23] B.R. Donald: *A Geometric Approach to Error Detection and Recovery for Robot Motion Planning with Uncertainty*, Artificial Intelligence. 37 pages 223-271, 1988.

[24] James Dugundji, *TOPOLOGY*, Allyn and Bacon, inc., Boston, Lonon, Sydney, Toronto, juillet 1976.

[25] Michel Erdmann: *Using Backprojections for Fine Motion Planning with Uncertainty*, The international Journal of Robotics Research, Vol. 5 No. 1, Spring 1986, Massachusetts Institute of Technology.

[26] E. Falkenauer and S. Bouffouix, *A Genetic Algorithm for Job Shop*, Proceedings of the 1991 IEEE International Conference on Robotics and Automation Sacramento, California, USA, avril 1991.

[27] E. Falkenauer, A. Delchambre: *A Genetic Algorithm for Bin Packing and Line Balancing* Proceedings of the 1992 IEEE International Conference on Robotics and Automation Nice, France - mai 1992.

[28] B. Faverjon : *Object level programming of industrial robots.* International Conference on Robotics and Automation, San Francisco California, USA, 1986.

[29] Bernard Faverjon, Pierre Tournassoud : *A Local Based Approach for Path Planning of Manipulators With a High Number of Degrees of Freedom*, Proceedings of the 1987 IEEE International Conference on Robotic and Automation, Raleigh, North Carolina USA, mars-avril 1987.

[30] Th. Freichard and C. Laugier: *Path-Velocity Decomposition Revised and Applied to Dynamic Trajectory Planning*Proceedings of the 1993 IEEE International Conference on Robotic and Automation, Atlanta, Georgia, USA - mai 1993.

[31] Davis E. Goldberg: *Genetic algorithms and machine learning*, University of Alabama, Tuscaloosa, University of Michigan, Ann Arbor, in Machine Learning, Kluwer Academic Publishers, 1988.

[32] David E. Goldberg: *Genetic algorithms in search, optimization and machine learning*, The University of Alabama, Addison-Wesley publishing company, inc. 1989.

[33] John J. Grefenstette: *Credit assignment in rule discovery systems based on genetic algorithms*, Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, D.C. in Machine Learning, Kluwer Academic Publishers, 1988.

[34] Gunter Rudolph : *Convergence Analysis of Canonical Genetic Algorithms*, IEEE Transactions on Neural Networks. Vol. 5, No. 1, janvier 1994.

[35] J.H. Holland: *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Pres, 1975.

[36] Yong K. Hwang and Narendra Ahuja: *Gross Motion Planning A Survey* Advanced Robotics Redundancy and Optimization, ACM Computing Surveys, Addison-Wesley Publishing Company Vol 24, No. 3, septembre 1992.

[37] Kenneth de Jong: *Learning with genetic algorithms*, Computer Science Department, George Mason University, in Machine Learning, Kluwer Academic Publishers, 1988.

[38] O. Khatib : *Real-time obstacle avoidance for manipulators and mobile robots.* Proceedings of the IEEE International Conference on Robotics and Automation, St. Lous Missouri, New York, U.S.A. 1985.

[39] D.E. Koditschek : *Robot planning and control via potential functions*, Robotics Review, Vol. 1 , Khatib, J. Graig, and T. Lozano-Pérez, Eds. MIT Press, Cambridge, Mass, 1989.

[40] Jean-Claude Latombe *Robot motion planning*, Kluwer Academic Publishers, Boston/Dordrecht/London.

[41] J.C. Latombe, A. Lazanas, S. Shekhar: *Robot motion planning with uncertainty in control and sensing*, Artificial Intelligence 52 pages 1-47, 1991.

[42] J-P Laumond, Siméon, T. R. Chatila, G. Giralt : *Trajectory planning and motion control for mobile robots*, IFAC/IUTAM Symposium on Dynamics of Controlled Mechanical Systems, Zürich, juin 1988, Proccedings, Springer Verlag.

[43] J-P Laumond, T. Siméon : Motion planning for a two degrees of freedom mobile robot with towing, IEEE ICON Conf., Jerusalem, avril 1989.

[44] Davis Lawrence (Editor): *Genetic algorithms and simulated annealing*, BBN Laboratoires, Cambridge, Massachusetts, 1987.

[45] Davis Lawrence (Editor): *Handbook of genetic algorithms* VNR Computer Lybrary, Van nostrand Reinhold Pub. New York, 1991.

[46] A. Lazanas, J.C. Latombe: *Landmark-Based Robot Navigation*, Proceedings of the Tenth National Conference on Artificial Intelligence. San Jose, CA, juillet 1992.

[47] A. Lazanas, J.C. Latombe: *Landmark-Based Robot Navigation*, Technical Report, Department of Computer Science, Stanford University May 1992.

[48] Tomás Lozano-Pérez, Matthew T. Mason and Russell H. Taylor: *Automatic Systhesis of Fine-Motion Strategies for Robots*, The international Journal of Robotics Research, Vol. 3 No. 1, Spring 1984, Massachusetts Institute of Technology.

[49] Tomás Lozano-Pérez: *A simple motion-planning algorithm for general robot manipulators*, IEEE Robotics and Automation, juin 1987.

[50] Tomás Lozano-Pérez and Patrick A. O'Donnell: *Parallel Robot Motion Planning*, Proceedings of the 1991 IEEE International Conference on Robotic and Automation Sacramento, California, USA, avril 1991.

[51] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, Patrick A. O'Donnell: *HANDEY : A Robot Task Planner*, Massachusetts Institute of Technology, U.S.A. 1992.

[52] M. Mäntylä *Introduction to Solid Modeling*, Helsinski University of Technology, Computer Science Press, Inc. Maryland, USA, 1988.

[53] M.T. Mason, *Automatic Planning to Fine Motions: Correctness and Completeness*, Proceedings IEEE International Conference on Robotic and Automation, Atlanta, GA, 1984.

[54] Emmanuel F. Mazer: *HANDEY: Un modèle de planificateur pour la programation automatique des robots*, thèse de doctorat de l'Institut National Polytechnique de Grenoble, Informatique, Décembre 1987.

[55] Emmanuel Mazer, Juan Manuel Ahuactzin, El-Ghazali Talbi, Pierre. Bessiere. *The Ariadnés Clew Algorithm*, From Animals to Animats: The Second International Conference on Simulation of Adaptive Behavior (SAB92). Honolulu, Hawaii, USA D'ecembre 7-11, 1992.

[56] Emmanuel Mazer, Juan Manuel Ahuactzin, El-Ghazali Talbi, Pierre. Bessiere. *Robot Motion Planning with the Ariadnés Clew Algorithm*, International Conference on Intelligent Autonomous Systems, Pittsburgh PA, USA, février 15-19, 1993.

[57] Emmanuel Mazer, Juan Manuel Ahuactzin, El-Ghazali Talbi, Pierre Bessière, Thierry Chatroux. *Parallel Motion Planning with The Ariadnés clew algorithm*, Third International Symposium On Experimental Robotics, Kyoto, Japan, 28-30 octobre 1993.

[58] Aimé Meygret, Martin D. Levine: *Extraction de primitives geometriques: Utilisation d'un algorithme génétique*, Rapport Annuel, Research Center for Intelligent Machines,McGill University, Montréal, Québec Canada. janvier 1992.

[59] Michel Monoux: *programation mathématique*, CNET,ENST, collection technique et scientifique des télécomunications, Ed. Dunod, Paris, France 1983.

[60] Mark H. Overmars: *A random approach to motion planning*, Utrecht University, Departament of Computer Science, The Netherlands 1992.

[61] Mark H. Overmars: *A random approach to motion planning*, Spring School on Robot Motion Planning, Rodez, France, mars-avril, 1993.

[62] Michel Pasquier: *Planification de trajectoires pour un robot manipulateur*, thèse de doctorat de l'Institut National Polytechnique de Grenoble, Informatique, janvier 1989.

[63] Judea Pearl : *HEURISTIQUE "stratégies de recherche intelligente pour la résolution de problémes par ordinateur"*, CEPADUES-EDITIONS, Toulouse 1990.

[64] C.B. Pettey, M.R. Leuze, J.J. Grefenstette, *A parallel genetic algorithm*, Proc. of the second Int. Conf. on Genetic algorithms, MIT, Cambridge, pp.155-161, Juil 1987.

[65] Franco P. Preparata, Michael Ian Shamos : *Computational Geometry : an intriduction*, Springer-Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest, 1985.

[66] M.B. Reid : *Path Planning Using Optically Computed Potential fields*, Proceedings IEEE International Conference on Robotic and Automation, Atlanta, mai 1993.

[67] Richard P. Paul : *Robot Manipulateurs mathematics, programming, and control*, The MIT Press, Cambridge, Massachusetts and London, England, 1981.

[68] G.Robertson, *Parallel implementation of genetic algorithms in a classifier system*, in Genetic algorithms an simulated annealing, L.Davis (editor), Morgan Kaufmann Pub., 1987.

[69] C.A. Rogers : *"Packing and Covering"*, Cambridge University Press, New York, 1964.

[70] Joseph O'Rourke, *Art Gallery Theorems and Algoritms*, Departament of Computer Science Johns Hoopkinns University. Oxford University Press. New York, Oxford.

[71] Alberto Rovetta, Remo Sala: *Robot Motion Planning With Parallel Systems*, Proceedings of the 1992 IEEE International Conference on Robotic and Automation Nice, France - mai 1992.

[72] Thomas L. Saaty: *Optimization in Integers and Related Extremal Problems*, McGraw Hill, USA, 1970.

[73] A. Schijver (Editor): *Pakink and Covering in Conbinatorics*, Mathematical Centre Tracts 106 p.141-177,.

[74] J.T. Schwartz, M. Sharir : *On the Piano Movers Problem I, II, III, IV, V"*. Courant Institut Technical Reports, Institut of Mathematical Sciences. New York University, 1983.

[75] J. Sanjiv Singh, Meghanad D. Wagh: *Robot path planning using intersecting convex shape: analysis and simulation*, IEEE Robotic and Automation avril 1987.

[76] Suk-Hwan Sus, Kang G. Shin: *A variational dynamic programming approach to robot-path planning with distance-safety criterion*, IEEE Robotic and Automation juin 1988.

[77] Petr Švestka :, *A Probabilistic Approach to Motion Planning for Car-Like Robots*, Utrecht University, Departament of Computer Science, The Netherlands 1992.

[78] Michel Taïx : *Planification de mouvements pour robot mobile non-holonome*, Thèse de l'Universite Paul Sabatier de Toulouse, Laboratoire d'Automatique et d'Analyse des Systèmes du CNRS, janvier 1991.

[79] Osamu Takahashi, R. J. Schilling: *Motion planning in plane using generalised Voroni diagrams*, IEEE Robotics and Automation, avril 1989.

[80] E-G. Talbi, P. Bessière: *A parallel genetic algorithm for graph partitioning problem*, Laboratoire de Génie Informatique/ Institut IMAG, 1991.

[81] E.G. Talbi, P. Bessiere: *A parallel Genetic Algorithm for the graph partioning problem*, ACM International Conference on Supercomputing, Cologne, juin 1991.

[82] E.G. Talbi, J.M. Ahuactzin, E. Mazer, P. Bessiere. *Parallel Robot Motion Planning*, CONPAR-92 - VAPP V, Lyon septembre 1992.

[83] E-G. Talbi, *Allocation de processus sur les architectures paralléles à mémoire distribuée*, Thèse de L'Institut National Polytechnique de Grenoble, Laboratoire de Génie Informatique/ Institut IMAG, Grenoble, France 11 mai 1993.

[84] R. Tanese, *Parallel genetic algorithm for a hypercube*, Proc. of the second Int. Conf. on Genetic algorithms, MIT, Cambridge, pp. 177-183, Juil 1987.

[85] P. Tournassoud *Géometrie et intelligence artificielle pour les robots* Hermes, Paris, France 1988.

[86] Charles W. Warren :*Fast Path Planning Using Modified A^* Method*, Proceedings of the 1993 IEEE International Conference on Robotic and Automation, Atlanta, Georgia, USA - mai 1993.

[87] Philippe Wenger :*Aptitud d'un robot manipulateur à parcourir son espace de travail en presence d'obstacles*, Thèse Robotique, Nantes 1989.