Planification de trajectoire robuste dans l'espace des informations capteur

Adnan Atassi

Université de Metz

U.F.R Sciences

THESE

présentée pour obtenir le titre de

DOCTEUR en SCIENCES de L'INGENIEUR

Spécialité

Automatique

par

ATASSI Adnan

Sujet de thèse :

PLANIFICATION DE TRAJECTOIRE ROBUSTE DANS L'ESPACE DES INFORMATIONS CAPTEUR

Composition du jury :

Mr Rachid ALAMI	Rapporteur
Mme Nadine Le FORT-PIAT	Rapporteur
Mr Alain PRUSKI	Directeur de thèse
	Examinateur
	Examinateur

Université de Metz

U.F.R Sciences

THESE

présentée pour obtenir le titre de

DOCTEUR en SCIENCES de L'INGENIEUR

Spécialité

Automatique

par

ATASSI Adnan

Sujet de thèse :

PLANIFICATION DE TRAJECTOIRE ROBUSTE DANS L'ESPACE DES INFORMATIONS CAPTEUR

Composition du jury :

Mr Rachid ALAMI	Rapporteur
Mme Nadine Le FORT-PIAT	Rapporteur
Mr Alain PRUSKI	Directeur de thèse
Mr Dominique MEIZEL	Examinateur
Mr Guy BOURHIS	Examinateur

A mes parents

A Mona et Akram

A Lara

Avant-Propos

Ce travail a été effectué au Laboratoire d'automatique des systèmes coopératifs (L.A.S.C.) de l'université de Metz dans le cadre du projet V.A.H.M. « Véhicules autonomes pour handicapés moteurs ».

Mes plus vifs remerciements vont à mon directeur de thèse Monsieur Alain PRUSKI, Professeur à l'université de Metz, qui m'a guidé durant ces années de thèse. Je le remercie de m'avoir accueilli au sein de son équipe et d'avoir partagé ses compétences scientifiques pour mener à bien cette thèse.

Je tiens à exprimer ma reconnaissance à Monsieur Rachid Alami, Directeur de recherche au C.N.R.S. et Madame Nadine Le Fort-Piat, Professeur à l'université de Besançon qui m'ont fait l'honneur de participer en tant que rapporteurs à ma soutenance de thèse.

Je remercie également Monsieur Dominique Meizel, Professeur à l'université de technologie de Compiègne, ainsi que Monsieur Guy Bourhis, Maître de conférence agrégé à l'université de Metz et faisant parti du L.A.S.C., pour avoir accepté de juger mon travail en participant au jury.

Je remercie enfin mes collégues du laboratoire qui m'ont soutenu durant ma thèse.

Sommaire

Listes des figures

Introduction générale

Introduction Générale

La robotique mobile a démarré en 1967, avec le robot « Shakey », réalisé par l'équipe du Stanford Research Institute [Nil 69]. Depuis, de nombreux travaux se sont succédés pour tendre vers un seul but : intelligence et autonomie des robots. Un robot intelligent est une machine capable de raisonner sur la tâche à accomplir et de mettre en œuvre une connexion intelligente entre perception et action. Le progrès en électronique, qui a permis de miniaturiser les circuits intégrés, a beaucoup contribué à l'évolution des robots intelligents. Une des étapes qui permet au robot d'évoluer dans l'environnement de manière autonome est le développement de planificateurs permettant au robot de concevoir de quelle manière il va réaliser la mission qu'on lui a confié.

De nos jours, il existe des domaines où les robots rendent d'immenses services à l'humanité. C'est le cas, lorsqu'ils permettent d'effectuer des tâches pénibles et répétitives (soudure, assemblage et transports de pièces, etc.), lorsqu'ils évoluent dans des environnements hostiles (dans le milieu marin, dans l'espace, dans un environnement irradié, dans un terrain miné, etc.), ou bien lorsqu'ils permettent de réaliser des activités nécessitant une précision extrême (microchirurgie, assemblage de précision, etc.).

Parmi les robots mobiles, certains sont conçus spécialement pour assister au déplacement des personnes handicapées dans leur vie quotidienne. Pour être opérationnels, ces robots font appel à plusieurs domaines de recherche de la robotique mobile :

- La modélisation (environnement, capteurs)
- La communication homme/machine
- La planification de trajectoire robuste
- La navigation et la perception
- L'intelligence artificielle
- La Commande

Les travaux présentés dans ce mémoire s'insèrent dans le cadre du projet « Véhicule Autonome pour Handicapé Moteur » (V. A. H. M.) développé au sein du Laboratoire d'Automatique des Systèmes Coopératifs (L.A.S.C.). Ce projet consiste à libérer une personne à mobilité très réduite des contraintes liées au déplacement d'un fauteuil électrique [Bour 93].

Les thèmes de recherche qui ont été abordés récemment au L.A.S.C. sont les suivants :

- Modélisation dynamique d'un environnement pour robot mobile à partir des données de localisation et de perception provenant de l'odométrie et des capteurs à ultrasons [Hab 94 et 95].
- Coopération homme-machine dans la conduite du fauteuil roulant robotisé : on cherche à utiliser au mieux les connaissances et compétences des deux entités coopérantes, la personne handicapée et la machine [Bour 98].
- Localisation statique d'un robot mobile dans un environnement intérieur, à partir du calcul par l'intermédiaire des capteurs à ultrasons de la position et de l'orientation du robot [Cou 98].
- Navigation d'un robot mobile : - modélisation de l'environnement par segment de droite, - élaboration d'une loi de commande robuste pour le suivi de chemin. Application au passage de portes [Nsi 98a et b].
- Architecture de commande du robot mobile : adaptative à tout changement de l'environnement, et générique pour tout ajout de nouvelles fonctionnalités du robot.
- Simulation graphique d'un système intelligent d'assistance à la conduite d'un fauteuil roulant électrique pour une personne présentant des déficiences motrices. Application à l'élaboration d'un modèle de comportement de la personne face à une classe de tâches à effectuer.

Dans ce mémoire, on propose une étude de faisabilité sur une nouvelle approche de la planification de trajectoire d'un robot mobile, basée entièrement sur les informations provenant des capteurs à ultrasons.

On cherche à limiter l'utilisation de l'odométrie qui introduit une accumulation d'erreurs lors du calcul de la position du robot. De ce fait, le robot ne sera plus localisé après un certain nombre de déplacements dans l'environnement. L'intérêt de cette thèse est de trouver des

chemins robustes par rapport aux erreurs de commande, du modèle de l'environnement et des informations capteurs.

Le chapitre I s'attachera essentiellement à décrire les différents types d'approches utilisées de nos jours pour planifier la trajectoire d'un robot, en distinguant les approches purement « géométriques » de celles qui tiennent compte de différents types d'incertitudes lors de la planification. Toutes ces approches consistent à trouver un modèle de l'environnement dans lequel s'effectuera à priori la planification avant de l'appliquer en temps réel.

Le chapitre II traite de tous les aspects de la modélisation de l'environnement du robot. D'abord, un modèle de capteur sera défini pour permettre de représenter l'observation d'un environnement par le robot, particulièrement sur les segments d'obstacles et d'en fournir les informations nécessaires pour la planification. Ensuite des régions seront représentées dans l'environnement, dans lesquelles le robot perçoit des informations distinctes sur les segments d'obstacles [Ata 98]. Enfin, un graphe de noeud est construit afin de mettre en relation les régions adjacentes dans lesquelles le robot perçoit un changement d'information.

Dans le chapitre III, une commande est appliquée au robot afin de pouvoir le faire évoluer d'une configuration initiale vers une configuration finale. Les vitesses linéaires et angulaires constituent les paramètres de sortie de la commande. A chaque instant, la commande a pour objectif de minimiser les écarts de mesures entre les informations mesurées en temps réel et celles que l'on désire obtenir dans le modèle à la configuration but. Cette approche est robuste par rapport aux erreurs du modèle de l'environnement et de la commande. Au niveau de la commande, aucune information sur la localisation n'est utilisée.

CHAPITRE 1

Etat de l'art

I. Etat de l'art

La planification en robotique est abordée sous deux points de vue: celui de la géométrie algorithmique et celui de la réalité physique:

- Dans le premier cas, on utilise une méthode géométrique de planification. Ceci implique que toutes les données du problème sont parfaitement connues, et que le problème lui même est défini avec exactitude. L'utilisation d'une méthode géométrique suppose que le robot est capable de contrôler tous ces déplacements, c'est à dire de suivre exactement le chemin géométrique généré par le planificateur. Cette hypothèse est réaliste lorsque l'espace est relativement peu contraint, que le robot est holonome, et que la configuration but ne doit pas être atteinte avec beaucoup de précision. Dans le cas où le robot est non holonome, les trajectoires générées par un planificateur géométrique ne peuvent pas toujours être suivis correctement. Dans la littérature sont présentées de nombreuses méthodes géométriques permettant de maîtriser la planification de trajectoires d'un robot mobile dans des environnements encombrés d'obstacles. Le lecteur pourra se référer à l'excellent ouvrage de J.C .Latombe qui fait une synthèse de l'ensemble de ces méthodes [Lat91]. On y trouve la méthode du graphe de visibilité, le diagramme de Voronoï, les méthodes par décomposition cellulaire et la méthode très populaire des potentiels.
- Dans le second cas, on utilise une méthode prenant en compte les incertitudes en position, sur les capteurs, sur la commande et sur le modèle de l'environnement utilisé. Cela consiste à trouver des chemins robustes depuis une configuration initiale jusqu'à atteindre une configuration finale en toute sécurité. Des commandes de déplacement, qui se basent sur la lecture de capteurs, sont utilisées afin de réduire ces incertitudes et de guider le robot vers son but. L'objectif est alors de générer des déplacements pour lesquels des caractéristiques spécifiques de l'environnement seront identifiables et détectables par les capteurs.

Dans ce qui suit vont être brièvement présentées des méthodes purement géométriques pour rappeler rapidement sur quels types de concept elles sont fondées. Ensuite, seront présentées plus en détail des travaux importants de ces dernières années sur la planification avec incertitudes.

1.1 - Planification géométrique

Dans ces méthodes, seul l'aspect géométrique du problème est pris en compte.

1.1.1 - Le graphe de visibilité

Cette méthode introduite par Nilsson [Nil 69] consiste à traduire la connectivité de l'espace libre par un réseau R généralement monodimensionnel.

Dans ce cas, la planification de chemins procède en deux temps: les configurations initiale (q_{init}) et finale (q_{goal}) sont d'abord raccordées au réseau R. Un algorithme de recherche de chemin dans un graphe appliqué à R permet ensuite de trouver un chemin reliant q_{init} à q_{goal}. L'idée de base de ce type d'approche consiste à réduire les dimensions de l'espace de recherche de manière à simplifier le problème de la planification. Les noeuds du graphe de visibilité sont constitués des sommets des obstacles polygonaux et des points initial et final. Un arc relie deux noeuds lorsque le segment constitué par ces deux noeuds n'intersecte aucun obstacle.

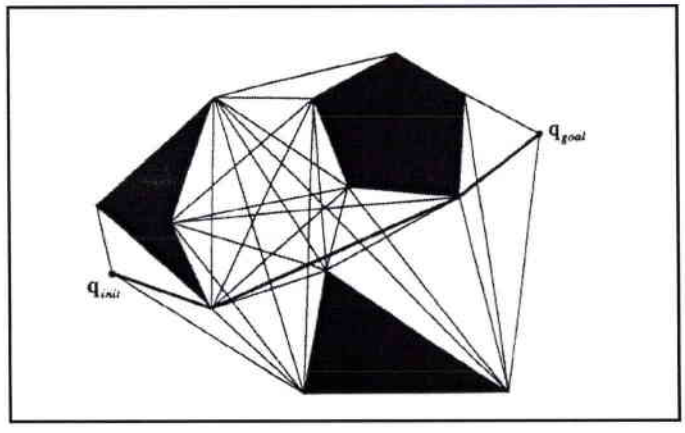


Figure 1.1
Exemple de graphe de visibilité extrait de [Lat 91]

1.1.2 - Diagramme de Voronoï

Cette méthode introduite dans le contexte de la robotique par O'Dunlaig et Yap [Dun 82] vise, comme la méthode précédente, à réduire d'une unité la dimension de l'espace de travail. Le travail de planification est ainsi grandement simplifiée, ce qui explique sa popularité [Pig94]. Cette méthode est caractérisée par la production de chemins qui passent au plus loin des obstacles: elle privilégie la sûreté d'un chemin. La planification se déroule en deux étapes: construction du diagramme, puis application d'un algorithme de recherche de chemins dans un graphe.

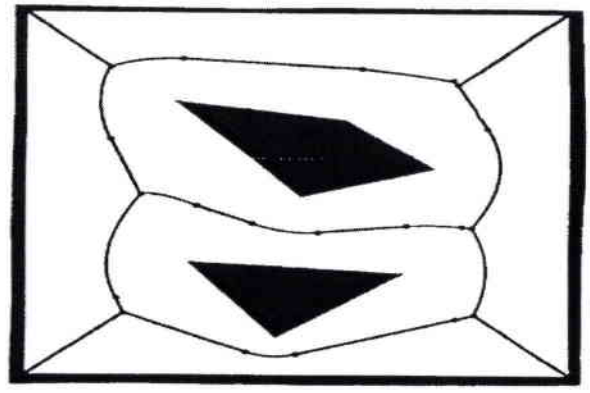


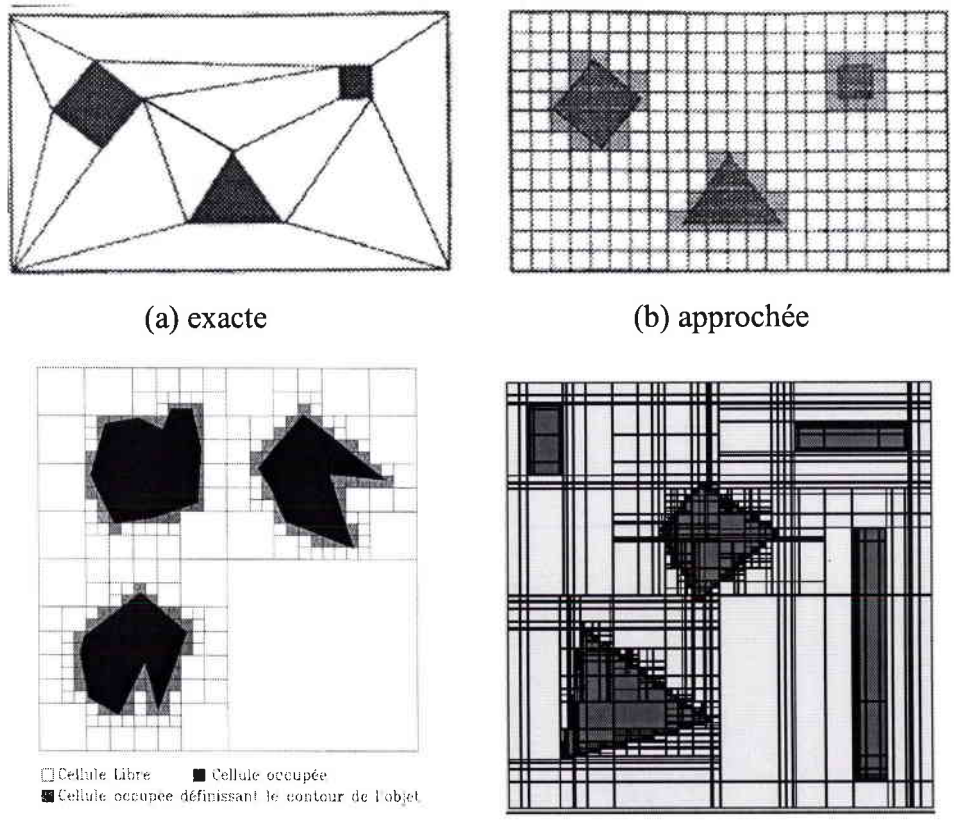
Figure 1.2
Exemple de Diagramme de Voronoï

1.1.3 - Décomposition cellulaire

Les méthodes de ce type, introduites par Lozano-Perez [Loz81] consistent à diviser l'espace libre en cellules. De manière générale, chacune de ces cellules est dite "libre" si elle ne contient aucune partie d'obstacle, ou "occupée" dans le cas contraire. L'espace libre est ainsi représenté par un graphe dans lequel chaque noeud représente une cellule libre, et chaque arc relie deux noeuds (quand les cellules correspondantes de l'espace libre sont adjacentes verticalement, horizontalement ou en diagonale).

Les méthodes de "décomposition cellulaire" se divisent en deux catégories: exacte et approchée. La décomposition cellulaire exacte [Kan 96] consiste à décomposer l'espace libre en un ensemble de cellules convexes le recouvrant totalement (figure 1.3). Ensuite, un noeud est placé au milieu de chacun de ces cellules. L'étape finale consiste à parcourir un graphe constitué de ces noeuds. Les chemins générés par cette méthode sont meilleurs que ceux

obtenues à partir d'un graphe de visibilité. Ils contournent les obstacles sans trop s'en éloigner. Les méthodes de type « approchée » ont plus de succès, notamment grâce à leur simplicité d'implantation. On y trouve la décomposition par quadtrees [Gil 66][Sam 80] qui correspond à la décomposition récursive de l'espace plan (2D) en un ensemble de cellules carrées de plus en plus petites, la décomposition par Octrees qui représente une décomposition dans un espace en trois dimensions [Jun 96][Pay 97] et le codage par NMV définie dans le chapitre suivant. Une fois le graphe de connectivité construit, un chemin reliant deux configurations données de l'espace libre est recherché au moyen d'un algorithme de recherche de type A* [Har 68].



(a) exacte (b) approchée

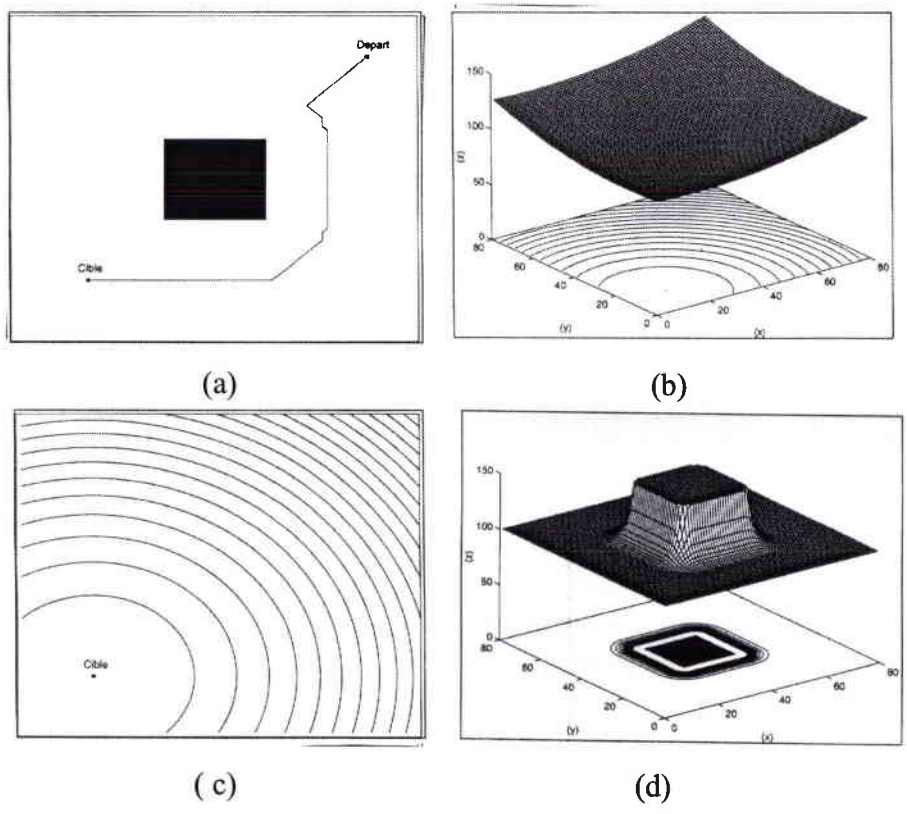
(c) Quadtrees (d) Codes multivaleurs

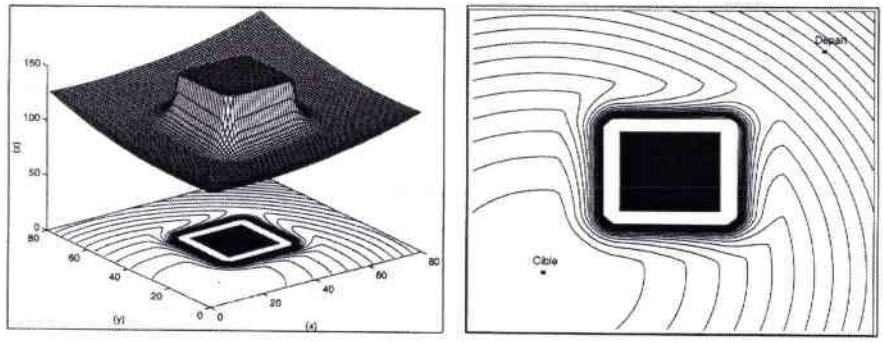
Figure 1.3 : Décomposition cellulaire exacte (a) et approchée (b)

1.1.4 - La méthode des potentiels

Cette méthode, introduite par Khatib [Kha 80] consiste à générer, aux alentours des obstacles, des champs de force fictives répulsives, provenant des obstacles, ainsi que des forces fictives attirantes vers l'objectif. Ces forces dérivent d'un champ de potentiels fictifs qui représente la somme des potentiels positifs et négatifs [Kha 86]. Afin de limiter l'influence des objets sur le robot, Khatib propose d'introduire une distance d'influence des objets sur le robot, au delà de laquelle les forces fictives de répulsion sont nulles. L'un des problèmes de cette méthode est la possibilité d'existence de minima locaux qui provoquent le blocage du robot. Khosla et Volpe proposent de réduire le nombre de ces minima en utilisant des fonctions de potentiels superquadratiques [Kho 88]. Un autre problème résulte de l'existence d'un ou plusieurs obstacles près du but: dans ce cas, le robot risque de s'éloigner, et on parle alors de rebondissement.

D'un point de vue pratique, la méthode des potentiels donne de bon résultats: elle permet par exemple de prendre en compte un obstacle imprévu pour l'éviter, et de le dévier en conséquence. Par contre, lorsque cette méthode est utilisée pour une planification globale, on a des effets secondaires comme des oscillations et des mouvements inconfortables.





(e) (f)

Figure 1.4

Construction de potentiels d'après Niniss [Nin 98]

a. Trajectoire obtenue par la méthode des potentiels
b. Potentiel attractif générée par le point cible
c. Equipotentielles associées au puits attractif.
d. Potentiel répulsif généré par un obstacle
e. Potentiel total résultant du point cible et de l'obstacle
f. Equipotentielles associées au potentiel total

1.1.5 - Conclusion

Dans toute méthode "géométrique", si une solution existe, elle peut être établie et servir de donnée de contrôle à un robot. Dans certains cas, cette solution aboutit effectivement à un plan réalisable par un robot physique (hors simulation). Mais dans d'autres cas, l'application pratique échoue, bien qu'une solution théorique existe. Les cas d'échec surviennent lorsque les incertitudes, qui n'ont pas été prises en considération, créent des perturbations qui modifient les données de base du problème.

1.2 - Incertitudes dans la planification

Les incertitudes en planification se situent à deux niveaux:

- Le modèle de l'environnement généralement considéré comme exact, provient d'une base de données CAO issue d'une prise de mesure humaine de l'environnement de travail. Ces mesures sont entachées d'erreurs sur la localisation et les dimensions des objets. Si nous supposons que les erreurs sur le modèle sont négligeables, nous ne sommes pas sûrs pour

autant de faire évoluer le robot vers un point but. Les informations issues du plan permettant au robot de se déplacer ne tiennent pas en compte des perturbations externes.

- La commande du robot s'effectue en général en boucle ouverte ou s'adapte en fonction des données de perception. De toute évidence, la précision de la boucle de commande dépend à la fois des informations issues du modèle, et des informations issues de l'environnement. Ce dernier constitue la deuxième source d'erreurs. Selon leur performance technologique, leur coût, leur nombre, leur zone d'action, leur résolution, les capteurs d'environnement introduisent des erreurs dont il faut tenir compte.

1.3 - Planification robuste

Une planification robuste consiste à considérer la présence des incertitudes dès la phase de la recherche de chemin. L'approche de la gestion des incertitudes se fait souvent en aval par l'élaboration d'un plan, à partir de données supposées exactes, puis en effectuant des mouvements en fonction de la réponse des capteurs, lors de l'exécution de la trajectoire. Cette méthode peut conduire à des situations de blocage.

En considérant les problèmes de non holonomie, et surtout en intégrant les capacités des capteurs d'environnement, il est nécessaire que le robot puisse acquérir suffisamment d'informations pour pouvoir tenir un raisonnement avant d'aborder certaines phases délicates de la trajectoire. La planification robuste concerne la prise en compte des contraintes physiques pour les mesures des capteurs, afin que les données qu'ils fournissent soient les plus exactes possibles.

Les capteurs d'ondes ultrasonores donnent des résultats de meilleure qualité, si l'obstacle est soit rugueux soit se situe perpendiculairement au faisceau. Ce type de planification dépend des performances et des contraintes des capteurs utilisés.

1.4 - Planification de trajectoire prenant en compte l'incertitude

Les premiers travaux de planification de chemin prenant en compte l'incertitude ont été effectués conjointement par Lozano-Perez [Loz 76] et Taylor [Tay 76]. La planification était différée, c'est à dire effectuée en deux phases : un premier chemin géométrique était généré sans prendre en compte l'incertitude, puis ce chemin était analysé afin d'obtenir une trajectoire robuste. Une telle approche a été reprise et améliorée en 1982 par Brooks [Bro 82]. L'intérêt d'une méthode différée réside dans sa relative simplicité qui lui permet de s'adapter aux robots ayant un grand nombre de degrés de liberté. Le principal inconvénient de l'approche est qu'il n'y a pas de garantie pour que dans toutes les situations, le chemin du robot ou un plan purement géométrique puisse être modifié localement pour prendre en compte les incertitudes. C'est pour cela, que des approches simultanées ont vu le jour, où l'incertitude est considéré dans le processus même de planification.

1.4.1 - Utilisation des préimages

1.4.1.1 - Déplacement du robot et contact avec l'environnement

Le déplacement du robot est défini par une information de vitesse v_c. Dans [Whi 77], l'auteur propose un modèle de commande compliant faisant intervenir une force f_c, exercée par le robot sur l'environnement, elle est proportionnelle à la vitesse de commande:

$$f_c = Bv_c$$

Dans l'espace libre, cette force est utilisée pour générer le mouvement. En contact avec un objet, ce dernier lui impose une force de réaction f_{react}. Les objets de l'environnement possèdent un coefficient de frottement Coulombien qui est représenté par un cône. Si la force se situe à l'intérieur du cône, la réaction annule totalement le mouvement: on se trouve alors en situation de collage. Si f_c est hors du cône, alors une force résultante f, entre f_c et les frottements, fait glisser le robot au contact (fig. 1.5).

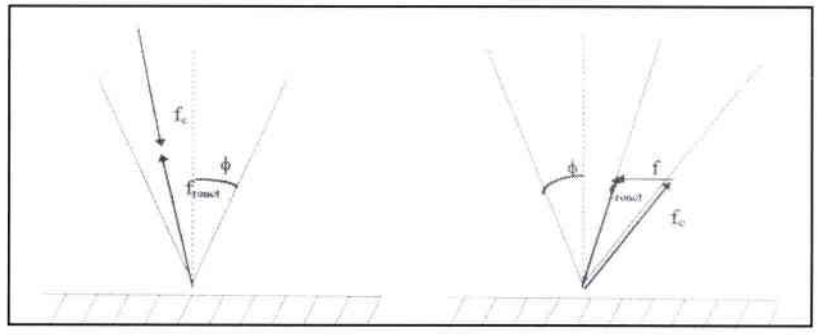


Figure 1.5 : Les situations de collage et de glissement.

L'incertitude sur la donnée de commande intervient au niveau de l'orientation de v_c. L'erreur est modélisée par un secteur angulaire centré autour du vecteur v_c, d'amplitude 2η (figure1.6).

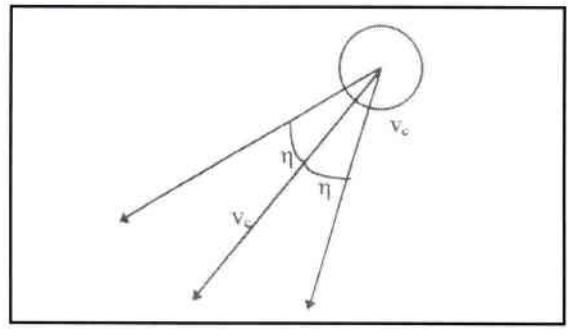


Figure 1.6 : L'erreur de commande

1.4.1.2 - Application des préimages

Etapes de préimage : Dans [Erd 86], l'auteur construit une préimage selon les étapes suivantes :

Etape 1: observant les sommets de l'environnement, on note tout sommet n'appartenant pas au but et pouvant engendrer un collage. Cette situation survient si le cône de frottement et le cône du vecteur de commande inversé, présentent une intersection non nulle lorsque leurs sommets sont confondus. On note aussi tout sommet but, à l'intersection d'une arête but et d'une arête non but, dont le collage est possible sur arête non but. On note enfin tout sommet but, à l'intersection d'une arête but et d'une arête non but, dont le mouvement est possible du sommet vers une arête qui n'appartient pas au but.

Etape 2 : placer le cône de commande inversée sur tout sommet noté. Déterminer les intersections de toutes les limites des cônes, entre eux, et avec les obstacles.

Etape 3 : créer la préimage en déterminant l'ensemble des arêtes des obstacles et des limites de cônes. L'énumération des arêtes de la préimage s'effectue à partir d'un sommet but dans le sens horaire, jusqu'à rencontrer le sommet but de départ.

Lorsque la région d'origine est incluse dans P_i, la trajectoire est alors définie par la séquence des vecteurs V_i associée aux préimages P_1, P_2, P_3, ... P_r, dans le sens inverse de construction.

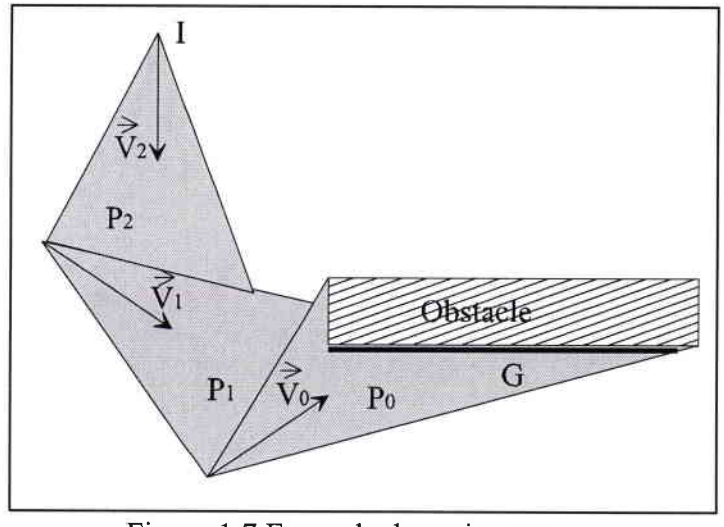


Figure 1.7 Exemple de preimages

L'aspect majeur de la méthode par préimage en chaînage arrière est sa généralité puisqu'elle peut s'appliquer à tout type de capteur et de contrôle. Cette méthode peut être utilisée aussi bien dans le domaine de la robotique mobile que dans celui des manipulateurs. L'application de cette méthode est néanmoins difficile à mettre en œuvre.

1.4.2 - Méthodes utilisant la projection à l'avant.

Cette approche, proposée par Laugier & al. [Naj 94] est issue du modèle de « backprojection » proposé par Erdmann [Erd 86]. A la différence de la méthode des preimages en chaînage arrière, cette approche procède en chaînage avant par "forward projection".

L'approche autorise et privilégie les contacts entre le robot et son environnement. Ces contacts sont utilisés pour réduire les incertitudes en position et en orientation du robot, et

pour le guider vers son but. L'incertitude de contrôle est prise en compte en introduisant un modèle géométrique évaluant « l'atteignabilité » des obstacles quand le robot se déplace dans l'espace libre.

1.4.2.1- Construction de la "forward projection" en tenant compte des incertitudes

La "forward projection"(projection en avant) consiste à identifier d'une part la région « atteignable » d'une surface d'un C-obstacle (obstacle grossi), et d'autre part l'ensemble des vitesses sûres permettant au robot d'atteindre son but de façon certaine, en dépit des incertitudes de position et de contrôle.

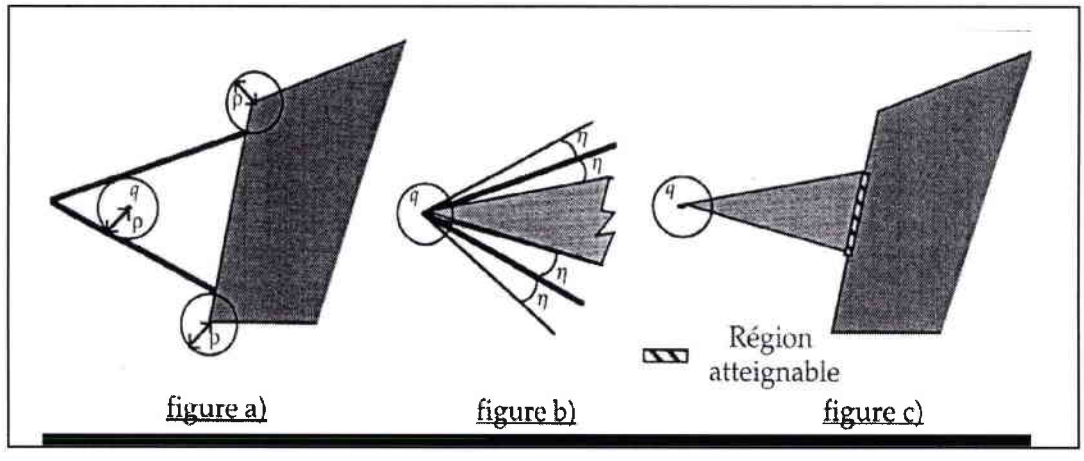


Figure 1.8 : Représentation de la projection en avant.

Etant donnée une incertitude de position modélisée par un disque de rayon ρ autour de la configuration q, et une incertitude de contrôle modélisée par un cône d'angle $2\eta<\pi$, une « forward projection » s'obtient comme suit:

(1) Pour éviter les contacts avec les sommets, on retranche, à chaque extrémité de la surface de l'obstacle, les segments à l'intérieur du cercle de rayon ρ (figure 1.8a),

(2) On trace les deux demi-droites tangentes au disque d'incertitude entourant q, et qui passent par les deux sommets déplacés. On obtient ainsi le cône des "vitesses potentielles"(fig.1.8 a).

(3) Pour éliminer l'incertitude, on rétrécit le cône en retranchant, de chaque limite du cône, les parties qui représentent les erreurs de commande η (représentées dans fig.1.6). Le cône résultant, s'il existe, est le cône des vitesses de sécurité (fig.1.8b).

(4) La projection du cône résultant (après élimination des incertitudes η) sur l'objectif, délimite la région atteignable.

1.4.2.2 - Procédure de planification

Deux fonctions sont utilisées pour la planification:

- une fonction de potentiel engendrant des mouvements continus du robot, et l'attirant vers des sous-buts.
- une fonction d'exploration définissant des sous-buts possibles en cas de minimum local. Des attracteurs locaux (balises) sont donc générés à l'image de l'algorithme du fil d'Ariane [Ahu 95].

1.4.2.3 - Utilisation de robots non holonomes

Dans [Frai 98], Fraichard et Mermond présentent un robot mobile qui prend en compte à la fois les contraintes de non holonomie et d'incertitude. Ils procèdent en deux étapes pour la planification. D'abord, ils utilisent un planificateur global, qui ne considère pas les contraintes de non holonomie et qui a comme fonction de construire un graphe reliant des noeuds situées dans des régions de localisation. Ensuite, un planificateur local génère des chemins de Dubins, en tenant compte de l'évitement d'obstacle.

1.4.2.4 - Exemples

Les figures suivantes donnent un exemple de résultat obtenu par cette méthode. Les petits cercles (fig1.9 et 1.10) représentent l'incertitude de contrôle qui tend à croître du fait de l'accumulation d'erreurs quand les mouvements ont lieu dans l'espace libre.

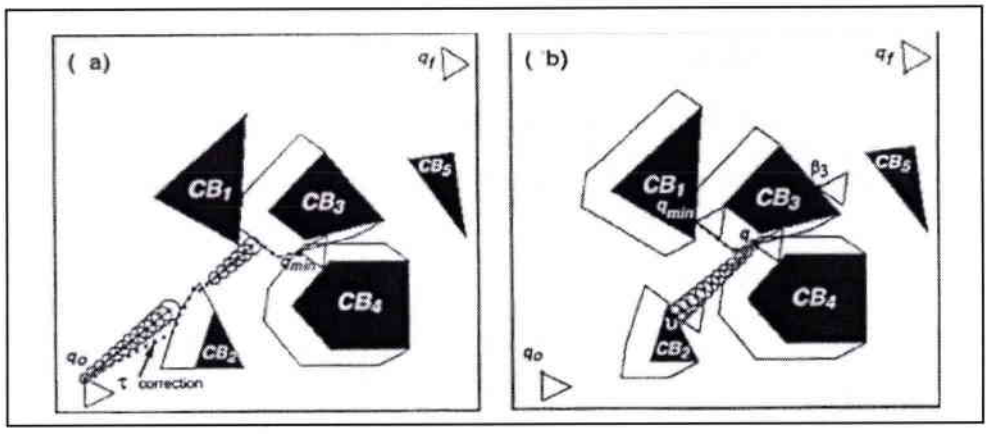


Figure 1.9 : Exemples de planification
avec minimum local extrait de [Naj 94]

1er cas (fig. 1.9): En tentant de rejoindre q_f, le robot, conduit par la fonction de potentiel, tombe dans un minimum local q_{min} (q_{min} étant une configuration de blocage)(fig.1.9a). La fonction d'exploration engendre deux attracteurs locaux, β_3 et ν (fig.1.9b). Malheureusement, voulant rejoindre β_3, la fonction de potentiel tombe dans un autre minimum local q_{min} (fig.1.9b). De plus, même si elle parvient à rejoindre l'attracteur ν, le but final ne peut être atteint directement à partir de cette configuration.

2eme cas (fig. 1.10): Dans cette figure, si la fonction d'exploration engendre un nouvel attracteur local (β_4) (fig. 1.10a) pouvant être atteint sans interférence de minimum local, le but final q_f est alors facilement atteint à partir de cette configuration β_4 (fig.1.10b).

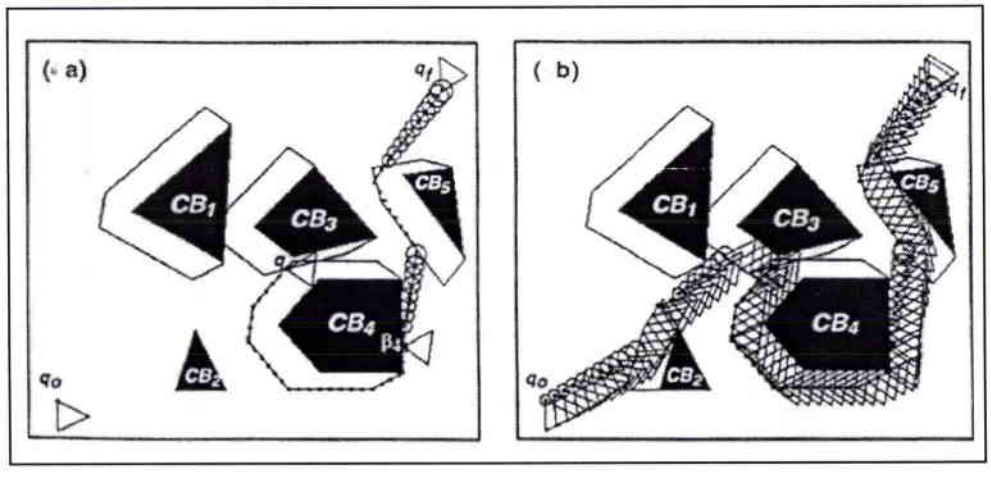


Figure 1.10 : Exemples de planification
sans minimum local extrait de [Naj 94]

1.4.3 - Méthode basée sur " la projection de l'incertitude "

Cette méthode se caractérise par des projections vers l'avant de l'incertitude de position du mobile. Décrite dans [Ala 94], elle consiste à projeter le point représentant le robot dans l'espace des configurations, selon un ensemble de directions possibles vers les obstacles. L'environnement est supposé parfaitement connu. Le robot est équipé d'un capteur de position ainsi que d'un capteur de contact. Dans [Bou 95], les auteurs utilisent la méthode pour un robot circulaire.

1.4.3.1 - Les primitives de commande.

Les déplacements du robot sont réalisés à partir de primitives dont la prise de référence s'effectue par contact avec les obstacles. Le robot peut évoluer de quatre manières différents, selon les quatre primitives suivantes :

- **MOVE_DISTANCE (θ,D):** Cette commande permet au mobile de se déplacer selon une orientation θ, d'une distance d.
- **MOVE_UNTIL_CONTACT (θ):** dans ce cas le robot se déplace vers un segment
- **FOLLOW_WALL_DISTANCES(D,[LEFT/RIGHT]:** la primitive consiste à déplacer le robot d'une distance d le long d'un segment, soit à droite, soit à gauche à partir d'un point donné sur ce segment.
- **FOLLOW_WALL_UNTIL_VERTEX([LEFT/RIGHT]):** La primitive consiste à déplacer le robot, le long d'un segment, soit à droite, soit à gauche, jusqu'à rencontrer un sommet. Comme les sommets des segments sont parfaitement connus, la détection d'un sommet réduit l'incertitude à zéro.

1.4.3.2 - Le planificateur

Le problème étudié consiste à relier un point P_I à un point P_G, tous deux appartenant à l'espace libre, en tenant compte d'une incertitude $\Delta\theta$ sur l'orientation du déplacement. Le planificateur développe un arbre de recherche en suivant une procédure particulière qui se base sur l'ensemble des cartes d'adjacence définies dans [Ala 94].

Chaque primitive va déterminer une distance de déplacement dont l'amplitude intervient dans le calcul du coût d'une fonction d'évaluation. Le développement d'un noeud de l'arbre

de recherche s'effectue en fonction du coût le plus faible de la fonction d'évaluation, à l'image d'un algorithme A*.

Si le point but P_G peut être rejoint directement à partir du point actuel (non initial) P par l'intermédiaire d'une primitive MOVE_DISTANCE, alors le mobile aura terminé sa trajectoire. Dans le cas contraire, la planification commence par annuler l'incertitude en générant une primitive FOLLOW_UNTILL_VERTEX, qui mène le robot vers l'une des deux extrémités du segment faisant obstacle. Puis pour chaque autre segment rencontré, on calcule la région adjacente pouvant être atteinte. Trois cas se distinguent :

- aucune région n'existe et aucun successeur n'est désigné. Le noeud est fermé.
- La localisation P est telle qu'une région adjacente existe. Un successeur est désigné pour les orientations extrêmes autorisées en P. Un mouvement selon la primitive MOVE_UNTIL_CONTACT est réalisé.
- aucune orientation ne permet de trouver une région adjacente. Un déplacement par la primitive FOLLOW_WALL_DISTANCES est activé jusqu'à un point P', sur lequel une adjacence peut être trouvée. En position P', une valeur de θ permettra d'activer la primitive MOVE_UNTIL_CONTACT pour générer un successeur.

La procedure est répétée jusqu'à ce que le point PG fasse partie d'un disque d'incertitude.

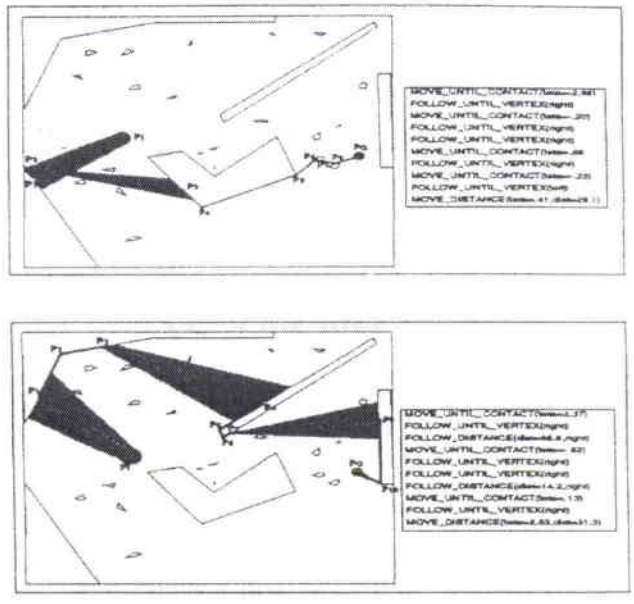


Figure 1.11 (extrait de [Ala 94])

Exemples de chemins adoptés en fonction des segments initiaux interceptés par le disque d'incertitude à partir du robot.

L'avantage de la méthode réside dans la génération de primitives de commandes robustes qui permet d'avoir une connexion entre le planificateur et le contrôle d'exécution simple. Cette méthode a néanmoins l'inconvénient de ne pas pouvoir trouver une solution dans le cas où aucun obstacle de l'espace n'est directement atteignable.

1.4.4 - Planification dans le champ d'incertitude d'un capteur.

Dans [Tak 92] et [Tak 94], les auteurs développent un modèle d'évaluation des incertitudes au niveau de la perception de l'environnement sur l'ensemble de l'espace de travail. Ils introduisent la notion de champ d'incertitude de perception ou SUF (Sensory Uncertainty Field). Celui-ci est défini de telle façon que les points de passage soient ceux dont les mesures permettant la déduction de la position soient les plus précises.

1.4.4.1 - Incertitude

Pour chaque configuration possible du robot, ce champ d'incertitude représente l'incertitude sur la configuration du robot calculée par une fonction de localisation, mettant en correspondance les données fournies par les capteurs et le modèle a priori de l'environnement. L'environnement étant supposé être composé d'obstacles polygonaux, le capteur percevra un ensemble de segments.

1.4.4.2 - Planification dans le champ de l'incertitude

Comme les auteurs utilisent une grille (décomposition cellulaire) pour la planification, le chemin est constitué par une séquence de configurations dans la grille. La planification consiste à trouver un chemin π sans collision entre une configuration source et une configuration but qui minimise la fonction :

$$C(\pi) = \sum \frac{1}{2}(F(q_i)^{\gamma} + F(q_{i+1})^{\gamma}) D(q_i, q_{i+1})$$

où F est une fonction qui mesure l'amplitude de la SUF, q_i et q_{i+1} les configurations à l'étape i et i+1, γ est paramètre qui permet de favoriser la fiabilité au dépend de la longueur du chemin, et $D(q_i, q_{i+1})$ est défini par :

$$D(q_i, q_{i+1}) = \max\left(\sqrt{(X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2}, \frac{1}{\mu}|\theta_{i+1} - \theta_i|\right)$$

où (X_i, Y_i) sont les positions du robot, et θ_i l'orientation du capteur.

La figure 1.12 représente les valeurs minimales de la SUF pour toutes les orientations θ du capteur, à chaque position (X,Y), dans une grille placée dans l'environnement. L'intensité en gris représente cette valeur. Plus le pixel est sombre, plus l'incertitude augmente. A chaque position du mobile, les petites flèches représentent l'orientation des capteurs qui délivrent l'incertitude minimale à cette position.

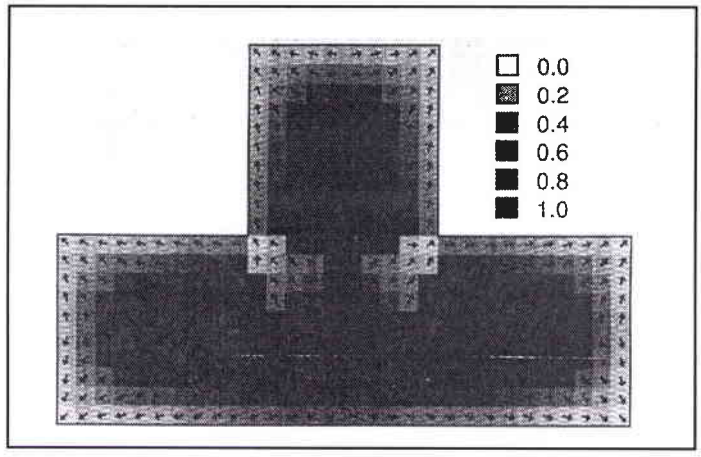


Figure 1.12 : Valeur minimale du SUF extrait de [Tak 94]

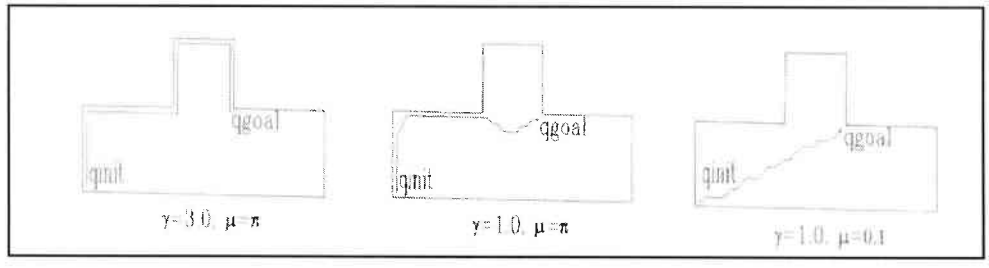


Figure 1.13 : exemples de trajectoires tirés de [Tak 94] pour plusieurs valeurs de μ et γ.

Le planificateur utilise le SUF pour générer des chemins qui minimisent les erreurs attendues en traversant des zones de l'espace de travail où les primitives de l'environnement sont visibles.

1.4.4.3 - Extension de la méthode

Vlassis and Tsanakas [Vla 98] étendent la notion du SUF à des environnements inconnues et non stationnaires. Les auteurs utilisent un modèle de réseaux de neurones structurés capables de trouver et de maintenir une estimation du SUF lorsque le robot se déplace dans l'espace libre. Le modèle est basé sur une localisation dynamique de l'environnement en utilisant les filtres de Kalman. Ils se servent de la triangulation de Delaunay pour réaliser une approximation polygonale du résultat donné par le SUF.

1.4.5 - Planification utilisant les cartes locales.

1.4.5.1 - Planification par " fonctions de tâche ".

Le planificateur est basé sur l'approche "fonctions de tâche" [Sam 91]. Il s'agit une réponse plus intégrée au problème de la robustesse de l'exécution de mouvements planifiés que la méthode précédente, par rapport à des imperfections de l'environnement vis a vis de son modèle. A partir d'un modèle de l'environnement et d'un modèle de capteur, des fonctions de tâche, à la fois compatibles avec les moyens de perception du robot, et directement exploitables par le contrôleur du véhicule [Cor 93], sont engendrées par le planificateur. Toute fonction de tâche est composée :

- d'un algorithme de contrôle en boucle fermée [Kana 91],[Sam 92], permettant de suivre un chemin nominal,
- d'événements qui enclenchent et déclenchent le fonctionnement de cet algorithme.

1.4.5.2 - Localisation et potentiel d'incertitude

Dans [Col 94] et [Col 95] Collin associe à chaque configuration q du robot une valeur scalaire positive appelée $CUP(q_o)$ (Configuration Uncertainty Potentiel), et une liste de primitives $L(q_o)$: $CUP(q_o)$ est un indicateur de l'incertitude de localisation du robot, obtenu par ses moyens de perception, quand sa configuration est dans le voisinage de q_o. $L(q_o)$ représente la liste de toutes les primitives utilisées par le véhicule pour se localiser.

Pour calculer la valeur CUP(q_o) ($q_o \in C_{libre}$) à une configuration q_o, on suit les étapes suivantes :

- **première étape** : elle consiste à déterminer la liste des couples (capteurs, primitive) capables de fournir une information fiable de localisation. En pratique, ceci est réalisé en simulant un balayage complet de l'environnement Il en résulte un ensemble **n** de mesures. Comme chaque mesure définit une bande de positions possibles du robot, on obtient **n** bandes dont l'intersection est appelée B (q_o)(fig.14).

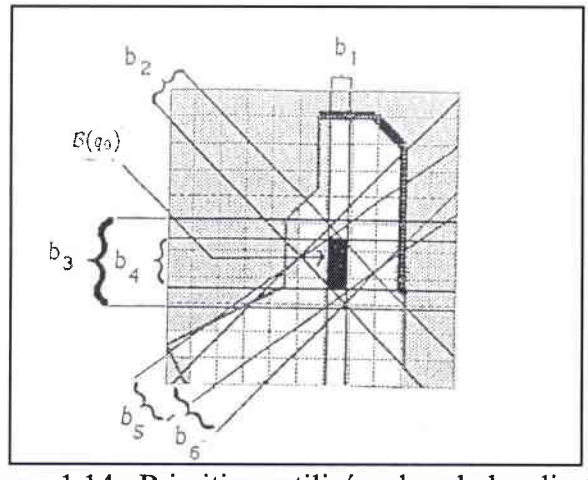


Figure 1.14 : Primitives utilisées dans la localisation

- **Seconde étape :** elle vise à simplifier la procédure de localisation du robot, en éliminant les bandes d'incertitude (mesures) non nécessaires à la définition de la surface de B(q_o). Il reste alors $N_m(q_o)$ mesures nécessaires et suffisantes pour définir la surface B(q_o) . L(q_o) est définie comme l'ensemble des $N_m(q_o)$ primitives utilisées par le véhicule pour réaliser ces $N_m(q_o)$ mesures.
- **Troisième étape :** elle consiste à mesurer la robustesse de la localisation par rapport à des déviations dans l'orientation du robot. En pratique, le robot est orienté dans un sens ou dans l'autre d'une petite valeur angulaire $\delta\theta$, qui mène à éliminer les mesures qui ne sont plus valides. Les $N_m(q_o)$ bandes d'incertitude sont alors légèrement modifiées, ce qui permet un nouveau ensemble admissible $B(q_1) \subset R$.

L'incrémentation de l'orientation θ du robot est répétée jusqu'à ce qu'après N_i incrémentations, au moins une parmi les $N_m(q_o)$ mesures ne soit plus valide. La même méthodologie est alors appliquée en décrémentant θ, donnant lieu à N_d décrémentations. A la fin de cette étape, un nombre (N_i + N_d +1) des surfaces admissibles suivantes sont définies :

Surf ($\beta(q_0)$), Surf($\beta(q_i)$) si ($1 \leq i \leq N_i$) et Surf ($\beta(q_{-j})$ si ($1 \leq j \leq N_d$). CUP(q_o) est alors définie comme suit :

$$CUP(q_0) = \frac{Surf(B(q_0)) + \sum_{i=1}^{Ni} Surf(B(q_i)) + \sum_{j=1}^{Nd} Surf(B(q_{-j}))}{1 + N_i + N_d}$$

Par cette définition, CUP(q_o) représente en quelque sorte la moyenne des surfaces admissibles dont la valeur q croit avec l'incertitude de localisation. Il est calculé en faisant référence aux primitives contenues dans L(q_o).

1.4.5.3 - Introduction des cartes locales dans la planification.

La planification de fonctions de tâche consiste à déplacer le robot à partir d'un algorithme en boucle fermée, en se servant de l'observation d'un ensemble de primitives faisant partie d'une carte appelée "carte locale". Une carte locale [Lam 94] doit contenir le minimum de primitives pour simplifier et augmenter la rapidité de l'algorithme de localisation.

A chaque carte locale L, les auteurs associent une structure locale F définie par deux axes de référence. Dès qu'une structure F est associée à L, une trajectoire nominale τ est calculée à partir de toutes les configurations référencées par rapport à L en utilisant les courbes de Béziers (figure 1.15).

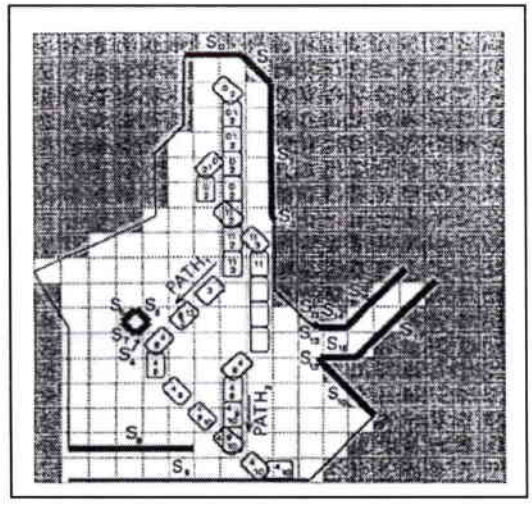


Figure 1.15 : Planification de trajectoire utilisant les cartes locales.

La planification de la trajectoire globale est constituée d'une séquence de sous trajectoires nominales, chacune étant définie dans sa propre carte locale. Pour avoir une planification liée

directement au contrôle de l'exécution, le robot va chercher chaque trajectoire en utilisant un algorithme de planification en boucle fermée pour le suivi de chemin. Un tel algorithme utilise la carte locale courante L, pour estimer la configuration du robot référencé F, qui constitue l'information de retour.

1.4.5.4 - Planification de tâches sûres.

Dans [Lam 98], l'auteur considère les incertitudes sur la configuration du robot, sur son contrôle et sur la position des amers. L'incertitude en configuration du robot représentée par un modèle probabiliste est calculée à partir d'un filtre de Kalman étendu opérant sur des données simulées. La rélocalisation du robot peut s'effectuer dans tout l'espace libre avec une précision plus ou moins importante. L'incertitude sur le contrôle est donnée par une formule analytique dont les variables sont les gains du contrôleur, ainsi que l'incertitude en configuration du robot. Enfin, l'incertitude sur la configuration des amers est traitée de manière implicite par l'utilisation de cartes locales. Les cartes locales sont représentées par un ensemble d'amers choisis destinés à la localisation.

L'approche utilisé par Lambert consiste à trouver continuellement un rapprochement entre la planification et le contrôle d'exécution. La détermination de l'incertitude sur le contrôle a permis de développer un planificateur de tâches sûres vis à vis des erreurs sur la position des amers. L'approche permet de planifier à la fois dans l'espace des tâches et des incertitudes, pour un robot non holonome, en utilisant un algorithme complet.

1.4.6 - Projection sur l'espace libre.

Dans [Roh 93] et [Pru 97], l'auteur décrit une méthode de planification robuste utilisant un modèle de l'espace des configurations admissibles, cet espace étant discrétisé par codage NMV (codage par nombres multivaleurs). Le principe de ce codage est décrit dans le chapitre suivant. L'objectif de cette méthode est de définir un chemin, en considérant les incertitudes au niveau de la commande. L'environnement est supposé connu. La méthode employée consiste à vérifier si la trace de l'ellipsoïde d'équiprobabilité des erreurs de position est incluse dans l'espace des configurations admissibles. Si cela n'est pas le cas (collision avec un obstacle), la trajectoire d'origine est déviée d'un coefficient donné (chemin localement modifié).

1.4.6.1 - Modèle de la trajectoire du robot

En se déplaçant d'un point X_{i-1} vers un point X_i, le robot se situera dans la trace laissée par l'ellipsoïde d'équiprobabilité qui permet de modéliser l'incertitude en configuration du robot (voir figure 1.16).

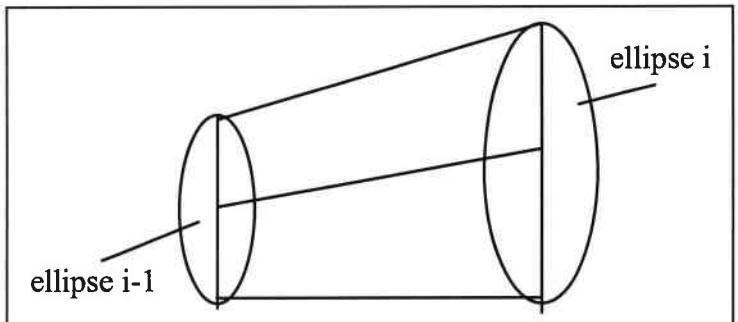


Figure 1.16 La trace de l'ellipsoïde

1.4.6.2 - Planification

La planification se déroule en deux phases: d'abord une planification est effectuée en ne considérant pas les erreurs dues aux incertitudes. Ensuite, une vérification de collision, entre la trace laissée par l'ellipse d'équiprobabilité et les obstacles du modèle, est réalisée. La vérification de la collision s'effectue à partir des codes des contours des obstacles.

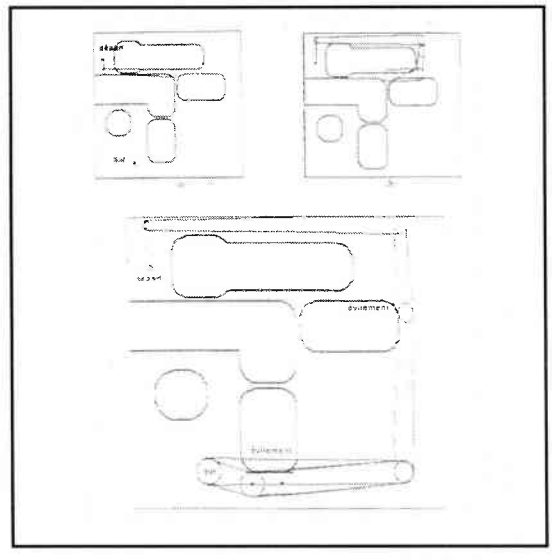


Figure 1.17 : Exemple de planification pour un robot circulaire

1.4.7 - Planification robuste de Page et Sanderson

Dans [Pag 95] et [Pag 94], les auteurs réalisent une planification de trajectoire robuste pour la commande, à base d'informations capteur, en tenant compte des incertitudes. Les algorithmes de planification emploient les capacités sensorielles d'un robot autant que nécessaire pour accomplir une tâche, en activant les capteurs ponctuellement ou pendant des intervalles de temps.

1.4.7.1 - Représentation du capteur

L'acuité des mesures d'un capteur dépend:

- D'un domaine, qui contient l'ensemble des configurations du robot, dans lesquelles des mesures valides peuvent être prises.
- D'un champ d'incertitude absolue des capteurs, qui détermine la précision globale du capteur.
- D'une incertitude en mouvement incrémental, qui détermine la précision relative du capteur, pertinente aux déplacements.

1.4.7.2 - Incertitudes

Trois types d'incertitudes sont prises en compte :

- Incertitude sur la commande (CU): elle décrit l'habileté du contrôleur à déplacer le robot vers l'endroit choisi.
- Incertitude sur les capteurs (SU): elle dépend non seulement de la configuration récente du robot, mais aussi des actions de mouvement et sensorielles qui ont fait aboutir à la configuration récente.
- Incertitude sur la position (PU): elle décrit la précision globale de la position du robot, lorsqu'une commande de position est appliquée.

1.4.7.3 - Algorithme de planification

Les auteurs utilisent deux types d'algorithmes complémentaires pour la planification de mouvement:

- le premier basé sur le chaînage arrière cherche globalement des moyens pour atteindre des sous-buts particuliers.

- L'autre cherche localement à satisfaire, dans un problème de planification, les contraintes au niveau des tâches, de l'incertitude et du domaine des capteurs.

Sur la figure 1.18, on peut voir une représentation du mouvement du robot basé sur les informations capteurs, pour passer de A à B. Un capteur de position est activé par défaut. Un capteur de proximité est ensuite activé de B à C, et enfin une caméra est activée au point E. Les lignes continues indiquent la position des robots et des obstacles, alors que les lignes en pointillé indiquent la position des obstacles grossis.

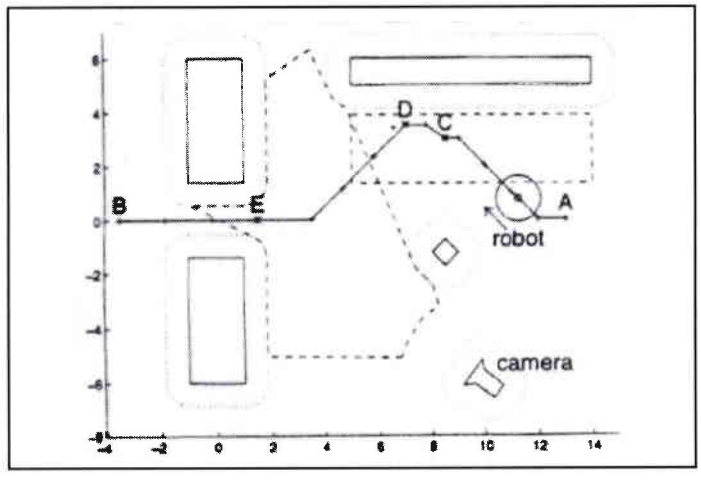


Figure 1.18 : mouvement du robot basé sur les informations capteurs.

Le problème principal de cette méthode est que l'incertitude en orientation n'est pas prise en compte. De plus, les trajectoires générées ne sont pas très satisfaisantes à cause de la différence d'informations obtenue entre la procédure de localisation et celle de planification.

1.4.8 - Planification par l'approche de " l'espace d'information ".

Cette approche utilise les programmations dynamiques stochastiques pour planifier la trajectoire du robot. Elle consiste à connaître l'état du robot à partir d'un vecteur d'informations provenant des capteurs.

En appliquant la méthode des DP (Dynamic Programming) stochastiques, les auteurs [Bar 95] développent des stratégies et des commandes de mouvement qui correspondent à une connaissance courante de la position du robot. L'exécution de ces stratégies et commandes assure au système les plus grandes probabilités d'atteindre une configuration but.

1.4.8.1 - Définition des DP stochastiques

La formulation et les principaux résultats des DP stochastiques sont présentés dans [Ber 87]. Les résultats sont basés sur le principe d'optimalité de Bellman [Bel 57].

L'objectif est de maximiser un coût, et obtenir ainsi le résultat désiré. Il faut cependant trouver un compromis entre le besoin actuel d'un coût élevé et la possibilité d'un coût final bas. Les auteurs supposent que le robot est contrôlé par un vecteur d'entrée u. La commande est sujette à des perturbations aléatoires w. La cinématique du robot est représentée par :

$$x_{k+1} = f(x_k, u_k, w_k),\ k = 0, 1, \ldots, N-1$$

où

- k est le nombre d'étapes entre deux configurations.
- x est la configuration du système à l'étape k.
- u est la commande sélectionnée à l'étape k.
- w est une perturbation de commande aléatoire.
- N est l'horizon du temps.

Les auteurs supposent que la définition de la cinématique prend en compte les contraintes sur l'évitement d'obstacles.

1.4.8.2 - Modèle du capteur

Les capteurs sont modélisés par l'observation de la forme:

$$z_0 = h_0(x_0, v_0)$$
$$z_k = h_k(x_k, u_{k-1}, v_k),\ k = 1, 2, \ldots, N.$$

où v_k est une perturbation aléatoire qui modélise les imperfections au niveau des capteurs, et h_k une fonction qui modélise la fonction interne des capteurs.

Si on dénote par I_k l'information valable à l'instant k, que les auteurs appellent "vecteur d'information", on aura:

$$I_0 = z_0$$
$$I_k = (z_0, u_0, z_1, u_1, z_2, ..., u_{k-1}, z_k),\ k = 1, 2, ..., N$$
$$I = (I_k, u_k, z_{k+1}),\ k = 0, 1, ..., N-1.$$

L'espace de tous les vecteurs d'information possibles à l'instant k est noté S^k et appelé "espace d'information".

1.4.8.3 - Planification

Le robot a pour mission d'optimiser une fonction de coût J. Cette fonction est additive dans la mesure où un coût $g_k(x_k, u_k, w_k)$ est ajouté à chaque instant k et un coût final $g_N(x_N)$ est ajouté à la fin du processus. Le coût total à travers chaque échantillon de trajectoire est :

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k).$$

La planification consiste à trouver des stratégies de mouvement $\pi = \{ \mu_0, \mu_1, \mu_2, ..., \mu_{N-1} \}$, qui offrent au robot une plus grande possibilité d'atteindre, en N étapes, une position but x_{but} à partir d'une position initiale x_{init}. Chaque fonction μ caractérise l'espace d'information S à travers l'espace de contrôle U. Etant donné un état initial I, il faut trouver une stratégie qui maximalise la fonction coût :

$$J_\pi(I_0) = E_0\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(I_k), w_k)\}.$$

E_k (k = 0, ..., N) dénote les espérances en fonction de l'information disponible à l'instant k.

1.4.9 - Planification utilisant les capteurs de vision

Dans [Miu 97], les auteurs proposent une méthode pour la planification de trajectoire en observant l'environnement des capteurs de vision. Les auteurs supposent que cette méthode peut être appliquée pour d'autres types de capteurs. L'information sur l'environnement est représentée par une distribution de probabilité multivariable, dans laquelle chaque variable est une propriété de l'environnement, comme par exemple une configuration du robot. Le planificateur sélectionne une multitude de points d'observation de telle manière que la réponse à l'attente du coût total, pour atteindre le but, soit minimale. Une formule récurrente est utilisée pour calculer le point d'observation suivant optimal du robot, ainsi que la condition d'observation suivante optimale, comme l'observation de la direction du robot, à partir de la position et de l'information sur l'environnement courante. Chaque résultat d'observation introduit une incertitude. Une prédiction de l'information est effectuée après chaque observation. Les solutions actuelles obtenues en utilisant une méthode qui permet d'obtenir le seuil en temps minimal, pour trouver le coût total, demande trop de calculs pour que ça soit applicable en temps réel.

1.5 - Conclusion

Les approches géométriques sont souvent nécessaires pour fournir un chemin nominal brut mais ne sont pas suffisantes. La stratégie utilisée dans ces approches ne permet pas de compenser les erreurs du modèle du robot et de l'environnement et conduit souvent à un échec de la mission. Les hypothèses considérées ne permettent pas à un véhicule de suivre précisément un chemin. Ces approches sont basées sur l'utilisation de capteurs proprioceptifs (odomètre, accéléromètre, etc.).

Même si le robot évolue dans un environnement connu, il est impossible de le situer précisément à l'aide de ce type de capteurs. En effet, ceux-ci ne font que fournir des renseignements relatifs à une localisation antérieure du robot. Ainsi à chaque nouvelle localisation, les erreurs des capteurs se cumulent aux erreurs précédentes. Un tel système de localisation présente l'inconvénient de fournir une estimée de la localisation du robot qui est de moins en moins précise au cours du temps.

De ce fait, il est nécessaire d'utiliser des capteurs extéroceptifs (télémètre laser, télémètre à ultrasons, etc.) permettant de fournir des informations sur la localisation relative du robot par

rapport à l'environnement. Bien que l'utilisation de capteurs de types différents soit un bon moyen de localiser précisément le robot, puisque cela permet de tirer partie de la qualité de chaque type, notre choix s'est orienté vers l'utilisation d'un seul type de capteur, les capteurs à ondes ultrasonores. En effet, ces capteurs sont très utilisés, pratiques, peu coûteux, et sont assez précis pour la détection d'obstacles.

Dans notre méthode, la planification de trajectoire tient compte de l'incertitude sur le modèle de l'environnement. Si en réalité la longueur d'un segment d'obstacle ou la distance entre deux obstacles sont différents de ceux qui ont été définies dans le modèle de l'environnement, le robot arrive à trouver un chemin au but. La commande du robot est robuste par rapport aux erreurs du modèle de l'environnement.

CHAPITRE II

Modélisation de l'environnement

Modélisation de l'environnement

Dans le cadre d'une stratégie globale de recherche de trajectoires pour robots mobiles, il est nécessaire de structurer de manière simple et rapide l'espace d'évolution du robot.

Le modèle dont peut disposer un robot lui permet de trouver un chemin dans la phase de planification, mais aussi de vérifier à tout moment si le chemin réellement suivi correspond à celui qui a été défini lors de la phase de planification.

Lorsque l'environnement est bien connu, le problème de localisation ou de recherche de chemin correspond à une opération de prédiction suivi d'une opération de vérification, qui n'est possible que si le robot est guidé par la prédiction des événements. Dans ce cas le robot est guidé par la détection, puis par l'analyse et l'évitement d'événements non attendus [Leo90]. Si une carte de l'environnement n'existe pas, alors la prédiction n'est pas possible et le robot ne peut se référer qu'à des informations issues de capteurs définissant des caractéristiques locales de l'environnement.

La modélisation de l'environnement permet d'avoir une connaissance a priori de l'environnement. Elle permet de disposer d'informations spécifiques sur le contenu de l'environnement afin de définir a priori une trajectoire possible. La modélisation sert aussi à comparer les informations acquises a priori sur l'environnement avec les informations des capteurs obtenues en temps réel. Les écarts entre ces informations servent ensuite à commander la trajectoire du robot.

La modélisation de l'environnement proposée consiste à créer dans l'espace libre des régions dans lesquelles le robot perçoit une ou plusieurs informations capteurs. Dans le cadre de ces travaux, l'environnement est constitué d'un ensemble d'obstacles polygonaux.

2.1 - Modèle de capteur

2.1.1 - Définition d'un modèle de capteur

Selon Leonard & Durant-Whyte [Leo 92], un modèle de capteur est une abstraction du traitement du capteur réel. Il décrit l'information qu'un capteur est capable de fournir, comment cette information est limitée par l'environnement, comment elle peut être enrichie par l'information provenant d'autres capteurs et comment elle peut être améliorée par une utilisation active du capteur physique.

Un modèle de capteur est composé de trois sous-modèles définis comme suit :

- un modèle d'observation, qui décrit les caractéristiques de la mesure du capteur
- un modèle de dépendance, qui décrit la communication d'information entre différents capteurs
- un modèle d'état, qui décrit comment les observations d'un capteur sont affectées par sa position

Cette définition considère l'information du capteur et la dépendance de cette information vis-à-vis de l'environnement.

2.1.2 - Intérêt de l'utilisation d'un modèle de capteur

L'intérêt de construire un modèle de capteur performant [Leo 92] réside essentiellement dans les deux capacités:

- à calculer a priori l'information que le capteur est susceptible de produire lorsqu'il scrute une scène bien précise à partir d'une position donnée.
- à reconnaître la géométrie de la scène dans laquelle est produite l'information étant donné un modèle d'observation de l'information du capteur.

Le modèle du capteur est l'élément clé sur lequel se base toute méthode de localisation ou de construction d'un modèle de l'environnement.

2.1.3 - Principe

Toute information acquise dans l'environnement du robot peut être utile pour le déplacement du robot. La représentation des informations capteurs par un ou plusieurs signaux est nécessaire pour les traiter et en tirer ceux qui contribueront par la suite à la planification de trajectoire du robot.

Dans un premier temps, nous supposons que le robot dispose d'un capteur télémétrique rotatif de plusieurs angles de mesure. D'abord, on considère les informations capteurs sans tenir compte des contraintes physiques sur les mesures des capteurs (angle de réflexion maximale autorisé pour une onde). Ensuite on intègre ces contraintes dans les mesures des capteurs pour que ça puisse correspondre au cas réel.

2.1.3.1 - Cas d'un segment d'obstacle infini

Les informations capteurs sont représentées par les orientations auxquelles sont effectuées les mesures du capteur par rapport à l'axe horizontal (référentiel global) au point A. (Voir figure 2.1)

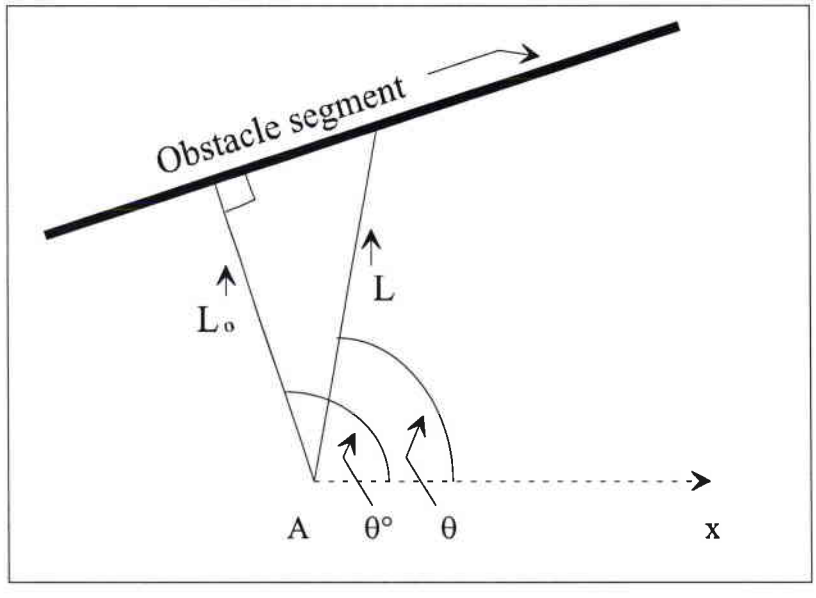


Figure 2.1 : Représentation des angles à des positions de mesure et des distances entre le robot et un segment d'obstacle.

A partir des distances entre le robot et un segment d'obstacle, pour plusieurs angles de mesure du capteur, et à partir de la connaissance d'un référentiel global, on peut connaître

l'orientation du segment d'obstacle par rapport à ce référentiel. La relation entre une mesure de la distance robot-obstacle, et entre l'orientation à laquelle est effectuée la mesure, est représentée par l'équation :

$$L = \frac{L_0}{\cos(\theta - \theta^\circ)}$$

où

- L_0 correspond à la distance minimal entre le robot et le segment d'obstacle.
- θ° représente l'angle à une position de mesure perpendiculaire au segment , c .à .d celui qui correspond à la distance la plus proche entre le robot et le segment.

On remarque que la fonction L tend vers l'infini lorsque θ tend vers $(\theta + \frac{\pi}{2})$ ou vers $(\theta - \frac{\pi}{2})$.

Pour obtenir une fonction (signal) finie et bornée, on prend l'inverse de la distance L. On obtient le signal suivant d(θ) :

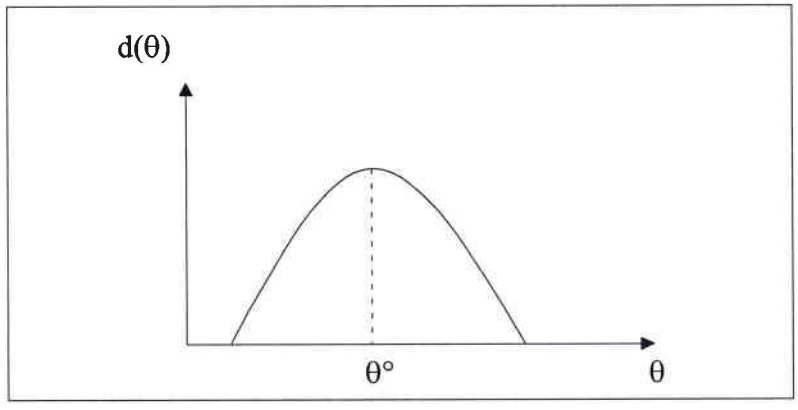


Figure 2.2 : Représentation du signal d(θ) avec un maximum pour θ = θ°.

On a alors la fonction suivante :

$$d(\theta) = \frac{1}{L} = \frac{\cos(\theta - \theta^\circ)}{L_0}$$

pour $\theta^\circ - \frac{\pi}{2} < \theta^\circ < \theta^\circ + \frac{\pi}{2}$ et d=0 dans le cas contraire.

Le maximum est atteint pour θ = θ°.

Soit d_m la distance maximale à laquelle le robot peut identifier un segment d'obstacle. On peut déterminer à partir du signal $d(\theta)$ et de d_m une région t_r dans laquelle le robot détecte à n'importe quel endroit ce segment.

2.1.3.2 - Cas où on a plusieurs segments

Dans le cas où on a plusieurs segments, si on considère s_1, s_2, ..., s_n l'ensemble des segments détectés par le robot, on obtient respectivement pour ces segments les signaux $d_1(\theta)$, ..., $d_n(\theta)$.

On considère deux cas :

- S'il n'y a aucune interférence entre ces signaux, la fonction globale sera :

$$d(\theta) = d_1(\theta) + d_2(\theta) + \ldots + d_n(\theta)$$

- Dans le cas d'interférence entre deux signaux $d_1(\theta)$ et $d_2(\theta)$ pour $\theta_1 < \theta < \theta_2$, la fonction résultante de ces deux fonctions est constituée du maximum entre les deux signaux :

$$d_r(\theta) = max(d_1(\theta), d_2(\theta)) \text{ pour } \theta_1 < \theta < \theta_2.$$

Max représente le maximum entre $d_1(\theta)$ et $d_2(\theta)$ pour $\theta_1 < \theta < \theta_2$.

Le signal global aura la forme suivante :

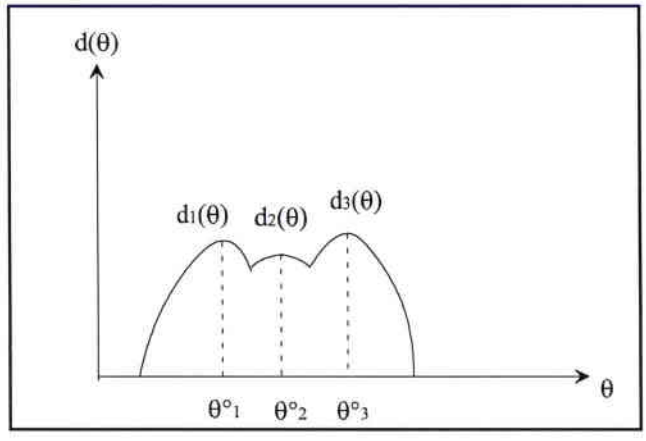


Figure 2.3 : Représentation de différents signaux avec interférence.

Les angles $\theta°_1$, $\theta°_2$, ..., $\theta°_n$ représentent respectivement les orientations à des positions de mesure du capteur, pour les distances minimales entre le robot et les segments s_1, s_2, ..., s_n (voir figure 2.4).

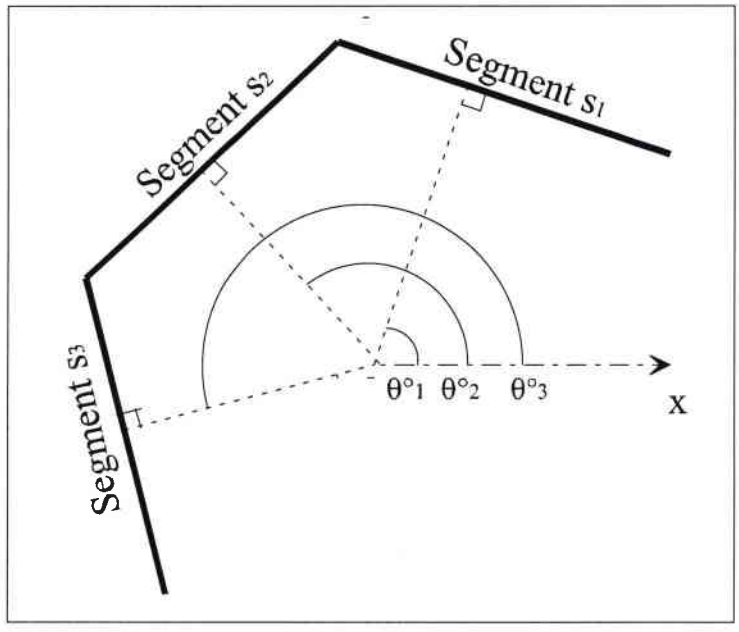


Figure 2.4 : Représentation des orientations
à plusieurs positions de mesure du capteur

2.1.4 - Représentation du signal capteur dans la simulation.

En simulation, les mesures du capteur rotatif sont obtenues à partir d'un lancer de rayon représentant l'onde émise du capteur, réalisé pour plusieurs angles de mesure. La variation entre deux angles adjacents est constante. On a choisit le nombre d'angles de mesure égal à 100.

2.1.4.1 - Principe du codage par NMV (nombres multivaleurs)

Pour pouvoir constituer le signal capteur à chaque fois qu'un obstacle est détecté à partir d'un lancer de rayon, on utilise un modèle de l'environnement. Ce modèle est basé sur l'utilisation des nombres multivaleurs.

Le codage par nombres multivaleurs consiste en un découpage des composantes de l'environnement (par exemple l'espace libre) au moyen d'un ensemble de rectangloïdes. Ces rectangloïdes ont la propriété de se recouvrir partiellement en fonction de leur configuration dans le modèle. Leurs dimensions sont fonction à la fois de leur localisation dans

l'environnement, et de l'espace disponible. La structure particulière du modèle implique que les dimensions de ces rectangloïdes sont proportionnelles à 2^n (n étant une variable entière).

Une des propriétés intéressantes des NMV est de permettre très rapidement par une fonction booléenne de vérifier l'intersection d'un point et d'un rectangloïde.

Pour avoir plus de renseignements sur les NMV, le lecteur peut se référer à [Pru 91], [Pru 92], et plus récemment à [Roh 93] et [Hab 95].

2.1.4.2 - Principe du lancer de rayon

Le lancer de rayon consiste à trouver, à l'aide du codage par NMV, le premier code rencontré faisant partie d'un obstacle, ceci en partant d'une position P_i connue, en direction d'un vecteur $\vec{u}$ donné. Cette procédure demande préalablement le calcul des paramètres d'une droite Δ, connaissant le vecteur directeur $\vec{u}$ et le point P_i. Il s'agit ensuite de déterminer à partir d'un algorithme de type *Bresenham* [Bre 65], l'ensemble ordonné des points qui constituent la droite en partant de P, dans la direction $\vec{u}$. (voir figure 2.5)

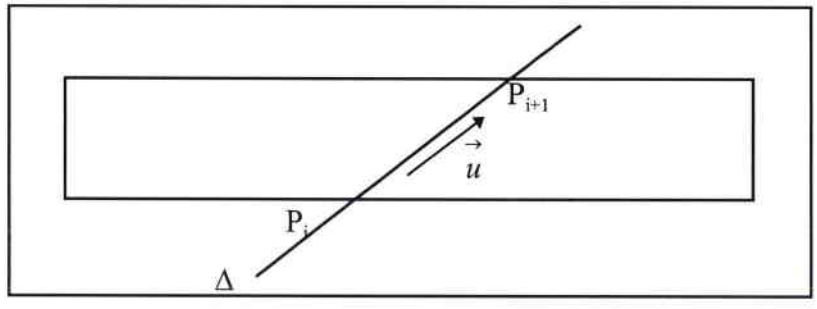


Figure 2.5 : Optimisation du lancer de rayon

On parcourt la droite en vérifiant, avec une fonction CHERCHE-CODE(), si chaque point fait partie d'une cellule libre ou occupée.

Si le code appartient à l'Espace Libre **Alors**
on prend le point suivant
Sinon Si le code appartient à L'Espace occupé **Alors**
le point considéré est le point résultat

On peut diminuer considérablement le nombre de points à vérifier en utilisant un des points d'intersection entre la droite et le code actuellement analysé. P_i étant le point étudié :

- Si P_i appartient à l'espace occupé, on s'arrête là.
- Si P_i est inclus dans un code appartenant à l'espace libre, alors le prochain point P_{i+1} sera le point d'intersection entre la droite Δ et les segments délimitant le code. Le point P_{i+1} est choisi de telle façon que les vecteurs $\overrightarrow{P_iP_{i+1}}$ et $\vec{u}$ aient le même sens et la même direction.

Cette procédure est particulièrement utile lorsque :

- Pour une configuration donnée du robot, on veut vérifier si les segments qui représentent les limites du robot entrent en collision avec un objet codé du modèle.
- On désire simuler un capteur à ultrasons au niveau d'un environnement modélisé.

2.1.4.3 - Balayage de l'environnement

Afin de pouvoir identifier tous les obstacles détectables autour du robot, on effectue un balayage de rayons complet autour du robot.

Le balayage est établi pour :

- une position de mesure $w_i \in [0,2\pi]$ exprimée en radian par rapport à un référentiel (axe horizontal).
- une distance maximum L_{max}. à laquelle le robot peut détecter un obstacle.

L'expression du point $P_i(P_{ix}, P_{iy})$ en coordonnées cartésiennes du segment d'obstacle détecté dans un code donné, en fonction du point $A(A_x, A_y)$ du robot est définie par :

$$P_{ix} = A_x + L_{max} \times \cos(w_i)$$
$$P_{iy} = A_y + L_{max} \times \sin(w_i)$$

Pour avoir le signal complet après le balayage de rayons, on utilise, pour chaque position de mesure w_i l'équation suivante :

$$d = \frac{1}{\sqrt{(P_i.x - A.x)^2 + (P_i.y - A.y)^2}}$$

d étant l'inverse de la distance entre le robot au point A et le point du segment d'obstacle P_i.

Le lancé de rayon est effectué lorsqu'un segment d'obstacle est détecté, en se servant du codage multivaleur défini précédemment, à une distance maximum de L_{max}.

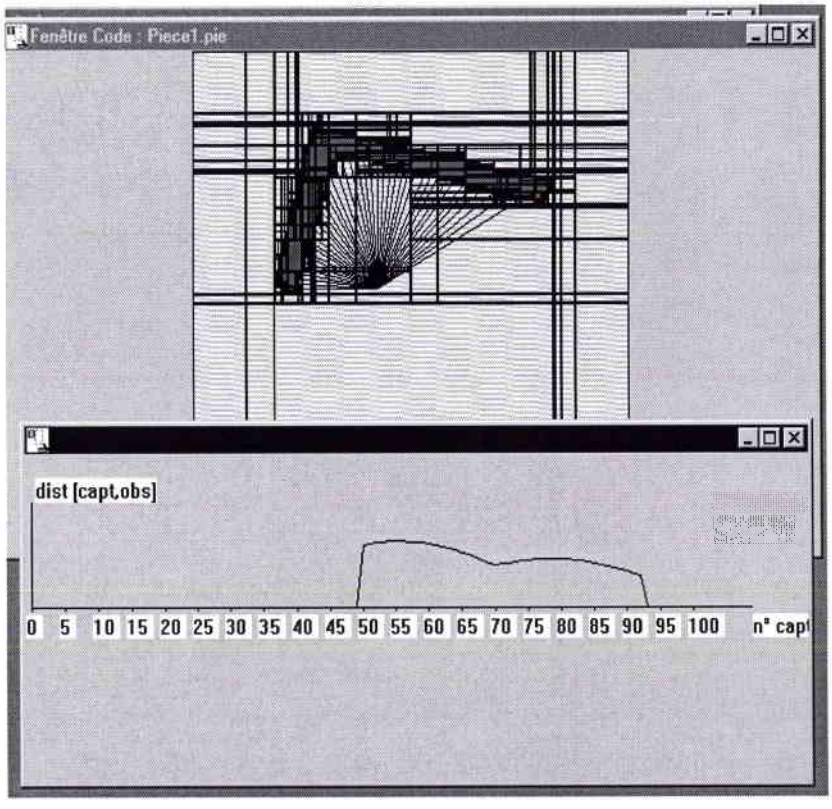


Figure 2.6

En haut : Représentation du lancer de rayons
lors de la détection de segments d'obstacle
En bas : Représentation du signal résultant
du lancer de rayons

La résolution du lancer de rayon est compris dans une fenêtre de dimension 256×256 pixels. On a un problème de précision du au nombre limité de pixels situés dans la fenêtre.

2.2 - Etude de capteurs utilisables dans le modèle

Avant de définir les diverses régions créées à partir des informations capteurs correspondant aux angles maximum des signaux $d_1(\theta),...,d_n(\theta)$, il serait intéressant d'expliciter les modalités de fonctionnement de deux types de capteurs : lasers et ultrasons

2.2.1- Capteur laser

Le capteur laser utilise une onde lumineuse. L'onde émise est constituée d'un faisceau étroit et très peu divergent qui permet une mesure très précise de la distance par rapport à un obstacle.

Pour pouvoir détecter un segment d'obstacle, le robot envoie un faisceau qui va réfléchir sur le plan du segment d'obstacle. Le faisceau retourne à sa source émettrice, à condition que l'angle entre la position de l'émetteur et le plan du segment d'obstacle où se produit la réflexion ne dépasse pas un angle critique θ_s. Dans le cas contraire, le faisceau diverge et est complètement perdu par le capteur. On n'aura alors aucune information sur le segment d'obstacle.(figure 2.7)

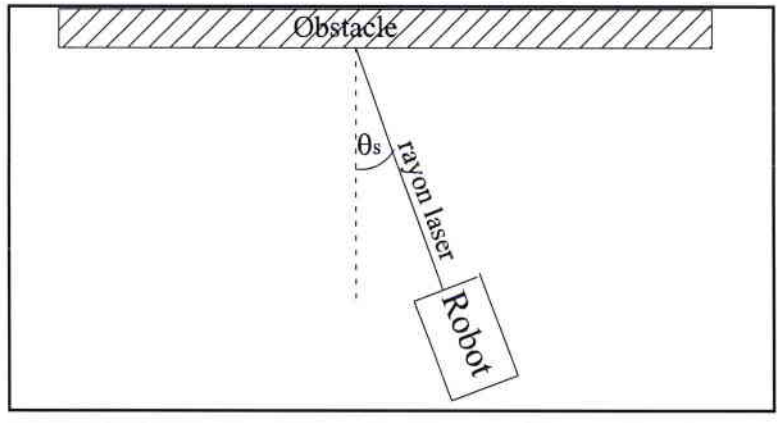


Figure 2.7 : Représentation du faisceau d'un capteur laser

L'inconvénient dans l'utilisation de ce genre de capteurs réside dans leur coût élevé, non recommandé pour le type d'application envisagé (robotique mobile pour handicapés). En effet, l'électronique qui commande ces capteurs doit mesurer un temps de vol très court (approximativement 3 ns par mètre parcouru). De ce fait la technologie utilisée est très coûteuse.

2.2.2 - Capteur à ondes ultrasonores

Les capteurs utilisant des ondes ultrasonores sont beaucoup plus abordables que le capteur laser, si on considère les aspects coût et complexité. En effet les temps de vol à mesurer sont beaucoup plus importants (approximativement 2 ms par mètre parcouru) et autorisent une électronique beaucoup moins sophistiquée. Par contre sa précision est faible à cause de leur grand cône d'ouverture, du temps de mesure qui n'est pas négligeable (environ 0.07s), et des mesures erronées qui peuvent apparaître à cause du phénomène de cross-talk. Il y a « cross-talk » lorsqu'une onde émise par un capteur est reçue par un autre, générant des impacts fantômes.

2.2.2.1 - Temps de vol

Le signal d'émission étant de durée limitée dans le temps, on mesure le temps qui sépare le top de réception, du top d'émission. Connaissant la vitesse du son, ce temps de vol correspond à une distance d (voir figure 2.8), où $d = \frac{T.V}{2}$. V étant la vitesse du son et T le temps de vol. Cette technique est dite de « l'écho pulsé ».

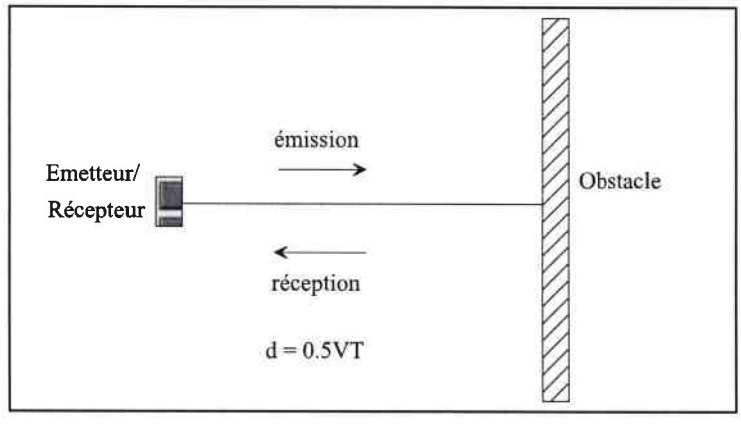


Figure 2.8 : Principe de la mesure d'un télémètre à ultrason.

2.2.2.2 - Environnement d'utilisation

L'utilisation d'un capteur ultrason est en général limitée à un environnement interne. En effet, la vitesse de l'onde ultrasonore est très sensible aux paramètres extérieurs du milieu de propagation, comme la température ou l'humidité. On peut toutefois remédier efficacement à ces inconvénients, c. à. d. réduire la sensibilité aux paramètres extérieurs, en effectuant des compensations sur les mesures obtenues.

2.2.2.3 - Description du transducteur ultrasonore.

Par définition, le transducteur ultrasonore constitue l'interface entre le milieu à étudier et l'électronique associée. Il est constitué d'un condensateur, dans lequel on exerce une différence de potentiel, qui crée une force électrostatique permettant d'avoir une variation de pression. Le transducteur peut fonctionner en émetteur comme en récepteur.

Dans sa phase d'émission, le transducteur, par variation de pression, transforme l'énergie électrique en énergie mécanique. L'émission d'une fréquence sonore est obtenue à partir de l'application d'une tension sinusoïdale aux bornes du transducteur. La fréquence d'émission est de 49.1 kHz. Cette fréquence correspond à la fréquence de résonance du circuit incluant le transducteur.

Dans sa phase de réception, si elle existe, le transducteur effectue la transformation inverse. L'énergie mécanique est transformée en énergie électrique par une variation de capacité.

2.2.2.4 - Cône d'émission d'un capteur à ultrasons

Un capteur ultrason émet dans un cône de $2\theta_s$' et détecte les segments ou les parties de segments d'obstacle qui se trouvent à l'intérieur de ce cône d'émission à la distance la plus courte (au point P sur la figure 2.9).

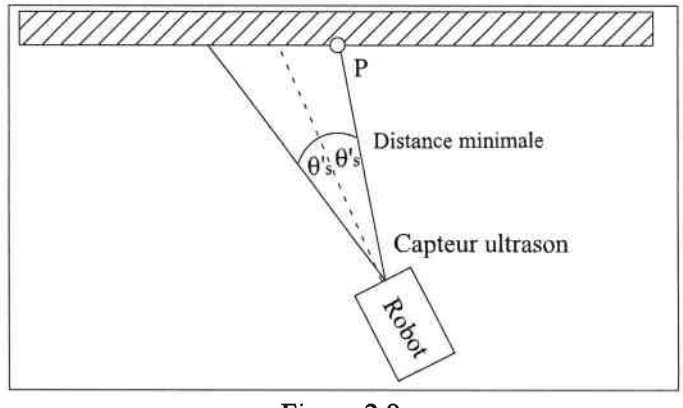


Figure 2.9

Détection d'un segment avec un capteur ultrason

Remarque :

On remarque que les deux types de capteurs (laser et ultrasons) acquièrent les mêmes informations lorsque $\theta_s = \theta_s$'.

2.2.3 - Prise en compte de l'angle de réflexion des capteurs

Soit α_{ref} l'angle de réflexion maximal du rayon émis par un capteur rotatif sur un segment d'obstacle. θ_{ref} est l'orientation de ce rayon par rapport à l'axe perpendiculaire au segment d'obstacle (voir figure 2.10).

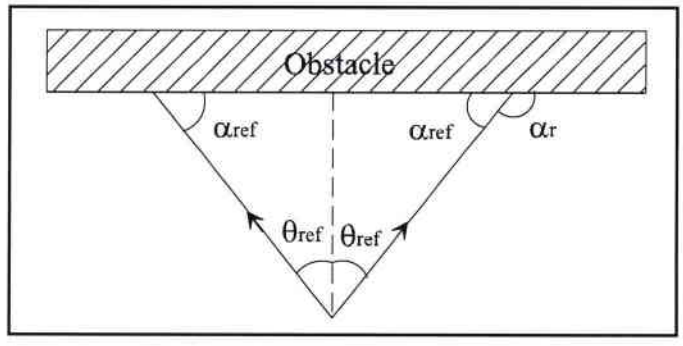


Figure 2.10

Représentation de l'angle de réflexion α_{ref} et θ_{ref}.

2.2.3.1 - Influence sur le signal

Si l'angle $\theta_{ref} < \frac{\pi}{2}$, alors le signal obtenu pour un segment sera écrêté à ses extrémités. (Voir figure 2.11). On obtient le signal suivant :

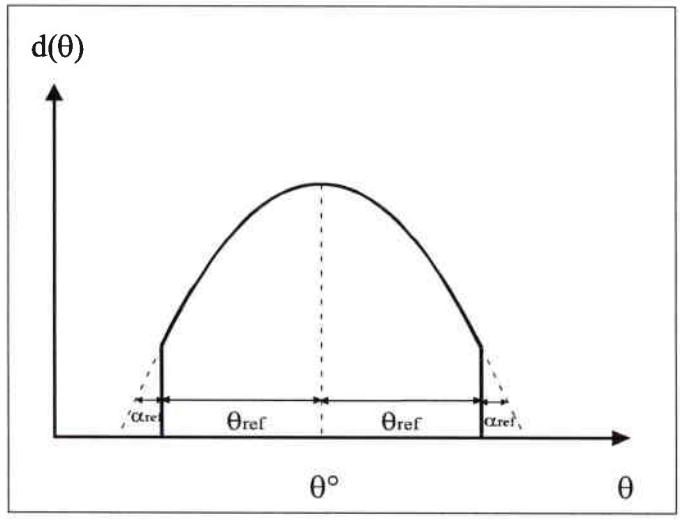


Figure 2.11 : Représentation du signal d(θ)
avec une réflexion α_{ref}.

Le changement du signal implique le changement de l'équation de la manière suivante :

$$d(\theta) = k.\cos(\theta°-\theta)$$

pour $\theta°-\theta_{ref} < \theta < \theta°+\theta_{ref}$.

$$d(\theta) = 0$$

pour $\theta < \theta°-\theta_{ref}$ et $\theta > \theta°+\theta_{ref}$.

2.2.3.2 - Influence sur le lancer de rayon en simulation

Pour que le lancer soit effectué en simulation sur un segment d'obstacle, il faudrait tenir compte de l'angle de réflexion maximale admissible sur ce segment, pour que le robot puisse recevoir en retour l'onde réfléchie.

Soit p_1 et p_2 les deux points d'un segment d'obstacle détecté par le robot au point A. A étant le point où le lancer de rayon est effectué et B le point du lancé de rayon appartenant au segment $[p_1,p_2]$ (voir figure 2.12).

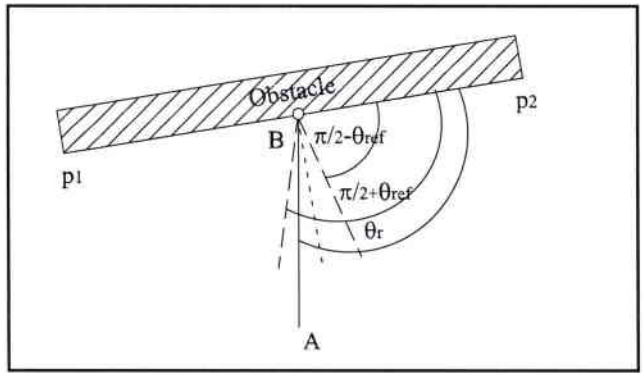


Figure 2.12

Condition de lancer de rayon en fonction
de l'angle de réflexion sur un segment d'obstacle.

On effectue le lancer de rayons sur la partie du segment qui vérifie la condition ci - dessus.

Pour que le robot au point A reçoive le point réfléchi, il faut que l'angle entre les segments [p1,p2] et [A,B] défini par θ_r soit tel que :

$$\frac{\pi}{2} - \theta_{ref} < \theta_r < \theta_{ref} + \frac{\pi}{2}$$

Dans ce qui suit, on va définir des régions représentant les informations (signaux) du modèle de l'environnement.

2.3 - Création de régions représentant les informations du modèle

L'objectif de cette partie est d'avoir une connaissance a priori, de toutes les zones dans lesquelles le robot perçoit un ou plusieurs segments d'obstacles de types ou en nombre différents. Cette connaissance nous permet de savoir a priori quels sont les types de segments qui sont détectables par les capteurs du robot à un endroit bien précis de l'environnement.

2.3.1 - Principe

On commence par identifier pour chacun des segments d'obstacles, la région dans laquelle le robot détecte le segment correspondant.

Soit t_r une région dans laquelle le robot détecte un segment d'obstacle.

- Si on ne tient pas compte de l'angle maximum de réflexion α_{ref} (ou de θ_{ref}) du rayon émis (cas d'un capteur laser), la région t_r aura la forme d'un rectangle de dimension d_m .l_s. l_s représente la longueur du segment d'obstacle associé à la région t_r. La région t_r est représentée sur la figure 2.13 par les points e_1, e_2, e_3 et e_4
- Si on prend compte de l'angle θ_{ref}, la région t_r va s'agrandir des deux cotés du rectangle de longueur d_m, par l'ajout respectif des sections angulaires de rayon d_m et d'angle θ_{ref}. (fig.2.13). On obtient alors une zone non polygonale (e_1, e_2, e'_4, e_4 ,e_3, e'_3), courbée en deux endroits. Pour simplifier cette zone, on prolonge le côté e_3 e_4 , jusqu'à obtenir des intersections avec les prolongements des côtés e_1 e'_3 et e_2 e'_4. On obtient alors une nouvelle région t_r trapézoïdale.

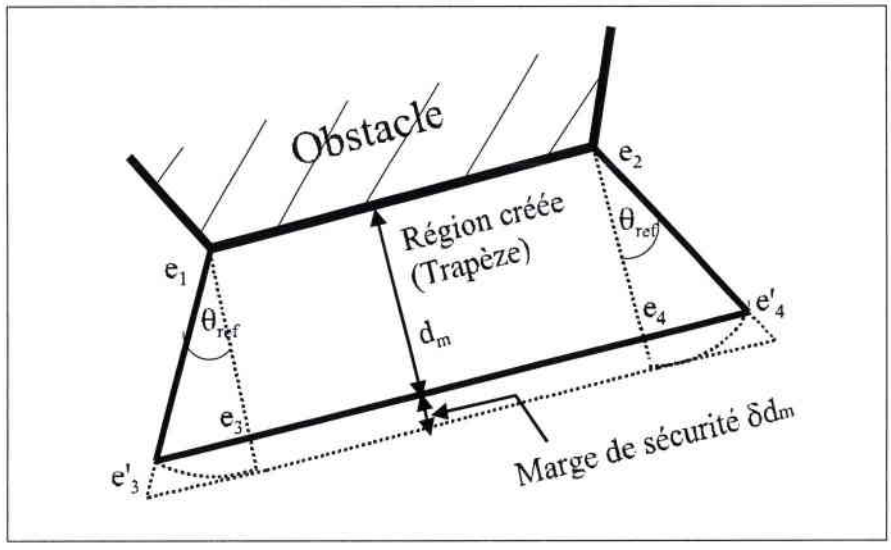


Figure 2.13

Représentation d'une région
où le robot peut détecter un segment d'obstacle

On estime que les erreurs qui se sont produites lors de la simplification de la zone t_r sont négligeables, d'autant plus que d_m (distance maximale à laquelle le robot peut détecter l'obstacle) est réduite par rapport à sa vraie valeur d'une marge de sécurité δd_m, ceci pour augmenter la garantie de détection.

Pour chaque segment d'obstacle, on associe une région trapézoïdale qui contiendra l'information sur son segment. On aura donc autant de trapèzes que de segments d'obstacles.

2.3.2 - Modélisation des régions sur l'ordinateur

On va montrer ici comment on modélise les régions trapézoïdales contenant les informations sur les segments d'obstacle, pour obtenir une carte de l'environnement.

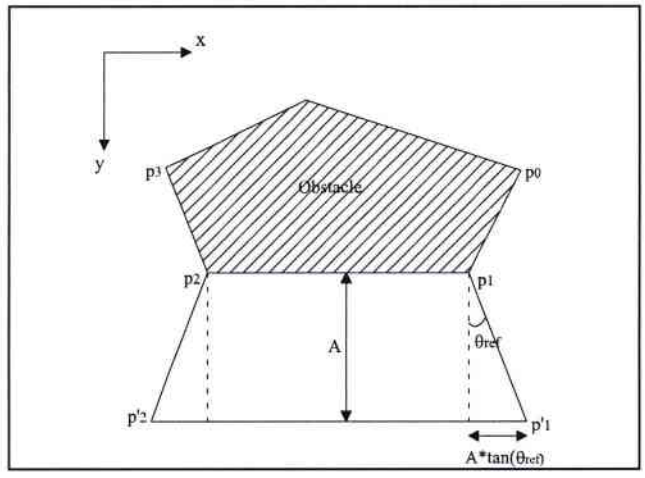


Figure 2.14 : Représentation d'un trapèze associé à un segment d'obstacle.

Pour représenter les trapèzes liés aux segments d'obstacles, on calcule les points des quatre côtés des trapèzes : Les deux points p_1 et p_2 du trapèze sont connus a priori, puisque ces points font parti des coordonnées de l'obstacle. Il reste à calculer les 2 points p'_1 et p'_2 en fonction respectivement des points p_1 et p_2.

On considère d'une manière générale que les coordonnées cartésiennes d'un point p_i selon les axes x et y sont définies respectivement par p_{ix} et p_{iy}.

On considère dans un premier temps que p_1 et p_2 ont des valeurs identiques selon l'axe des y, c'est à dire que $p_{1y} = p_{2y}$. Dans ce cas les valeurs de $p'_1(p'_{1x}, p'_{1y})$ et $p'_2(p'_{2x}, p'_{2y})$ sont définies en valeurs cartésiennes par :

$$p'_{1x} = p_{1x} - (A.\tan(\theta_{ref}))$$
$$p'_{1y} = p_{1y} + A$$

$$p'_{2x} = p_{2x} + (A.\tan(\theta_{ref}))$$
$$p'_{2y} = p_{2y} + A$$

où A représente la largeur du trapèze

Si le segment $[p_1, p_2]$ du trapèze est orienté d'un angle θ_1 par rapport à l'axe des x ($p_{2y} \neq p_{1y}$), on est amené, dans ce cas, à faire un changement de coordonnées :

- On calcule d'abord les valeurs cartésiennes de $p'_1(p'_{1x}, p'_{1y})$ et $p'_2(p'_{2x}, p'_{2y})$ par rapport aux axes x' et y'(fig. 2.15). Pour p'_1 on a :

$$p'_{1x'} = \frac{A}{\cos\theta_{ref}} \cos\theta_2 + p_{1x'}$$

$$p'_{1y'} = \frac{A}{\cos\theta_{ref}} \sin\theta_2 + p_{1y'}$$

avec θ_2 l'orientation du segment $[p_1, p'_1]$ par rapport à l'axe x

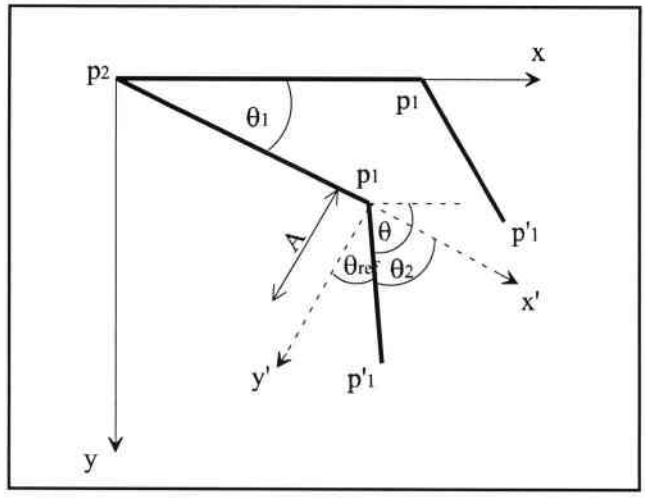


Figure 2.15 : Représentation de l'orientation d'un trapèze par rapport au référentiel global.

- On calcule ensuite les valeurs cartésiennes de $p'_1(p'_{1x}, p'_{1y})$ et $p'_2(p'_{2x}, p'_{2y})$ par rapport aux axes x et y. On obtient pour p'_1 :

$$p'_{1x} = \frac{A}{\cos\theta_{ref}} \cos\theta + p_{1x}$$

$$p'_{1y} = \frac{A}{\cos\theta_{ref}} \sin\theta + p_{1y}$$

avec $\theta = \theta_1 + \theta_2.(\theta_2 = \pi - \theta_{ref})$

En développant les équations ,on obtient :

$$p'_{1x} = -A.\tan(\theta_{ref}).co + A.si + p_{1x}$$
$$p'_{1y} = -A.\tan(\theta_{ref}).si - A.co + p_{1y}$$

$$p'_{2x} = A.\tan(\theta_{ref}).co + A.si + p_{2x}$$
$$p'_{2y} = A.\tan(\theta_{ref}).si - A.co + p_{2y}$$

où

- $$co = \cos\theta_1 = \frac{P_{2x} - P_{1x}}{\sqrt{(P_{2x} - P_{1x})^2 + (P_{2y} - P_{1y})^2}}$$
- $$si = \sin\theta_1 = \frac{P_{2y} - P_{1y}}{\sqrt{(P_{2x} - P_{1x})^2 + (P_{2y} - P_{1y})^2}}$$

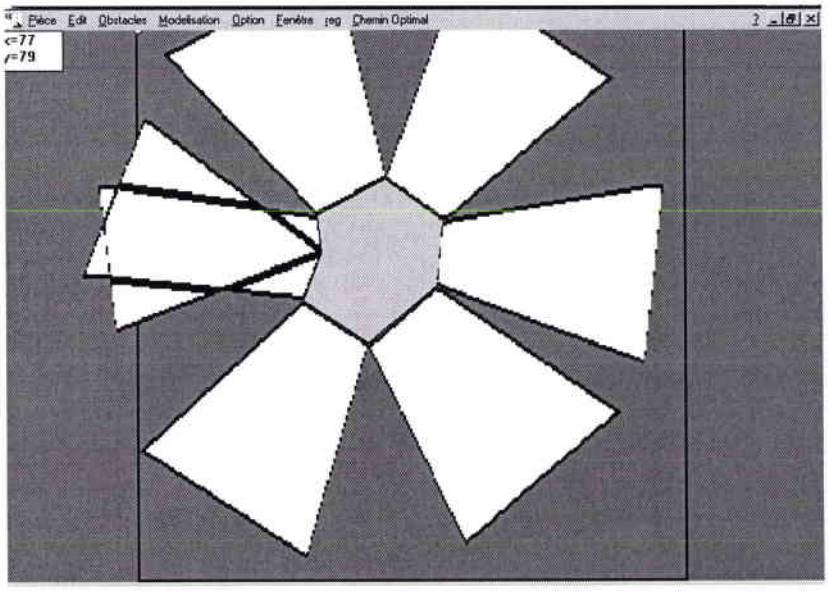


Figure 2.16 : Représentation d'un polygone (en gris clair) avec tous les trapèzes associés à ses segments(en blanc).

2.3.3 - Obstruction des parties de régions sans information capteur

Dans ce qu'on a vu précédemment, les informations sur les segments d'obstacle sont obtenues dans les régions trapézoïdales associées à ces segments. Ceci est valable si aucun obstacle ne traverse, ou ne coupe en deux, un trapèze représentant l'information sur le segment d'un autre obstacle (Figures 2.17a et 2.17b).

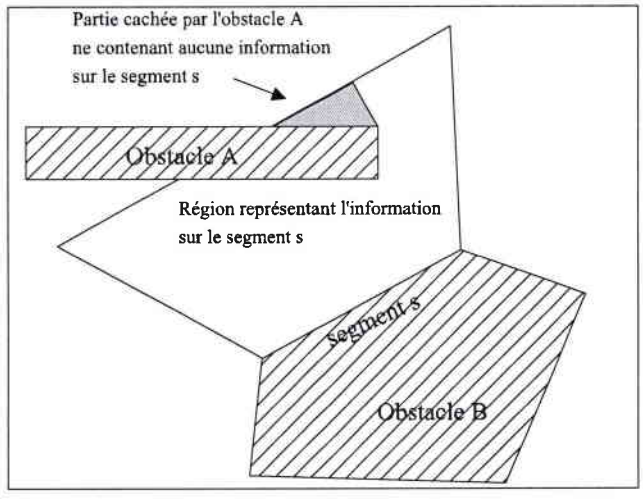


Figure 2.17a

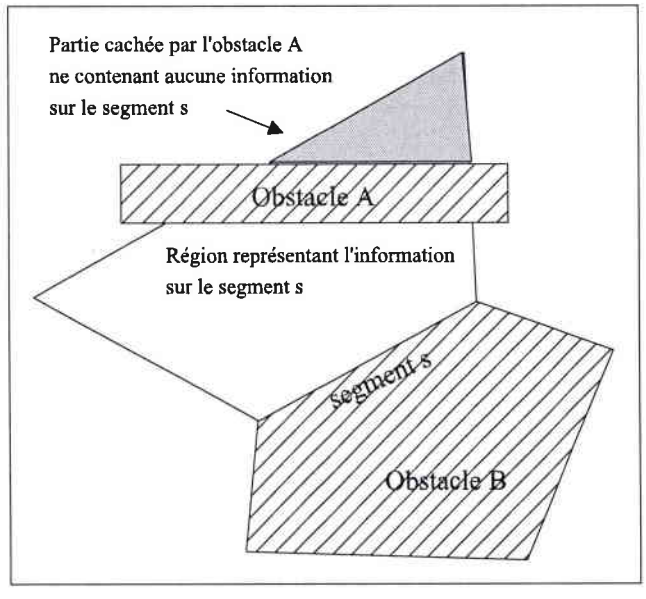


Figure 2.17b

Exemple d'obstacles qui traversent (Fig. 2.17a)
et qui coupent(fig.2.17b) un trapèze

Dans le cas de la fig. 2.17, où un obstacle traverse un trapèze, Il faut enlever les parties du trapèze cachées par l'obstacle dans lesquelles le robot ne peut pas détecter le segment associé au trapèze.

2.3.3.1 - Définition

- Soit Pt_1,Pt_2,Pt_3 et Pt_4 les quatre points d'un trapèze associé à un segment d'obstacle.
- Soit P_g et P_d les points extrêmes (droite et gauche) d'un autre obstacle situés à l'intérieur ou sur les bords du trapèze.
- et soit θ_g et θ_d les angles respectivement entre les segments [P_g,Pt_1] et [Pt_1,Pt_2], et entre [Pt_1,Pt_2] et [Pt_2,P_d] (fig.2.18).

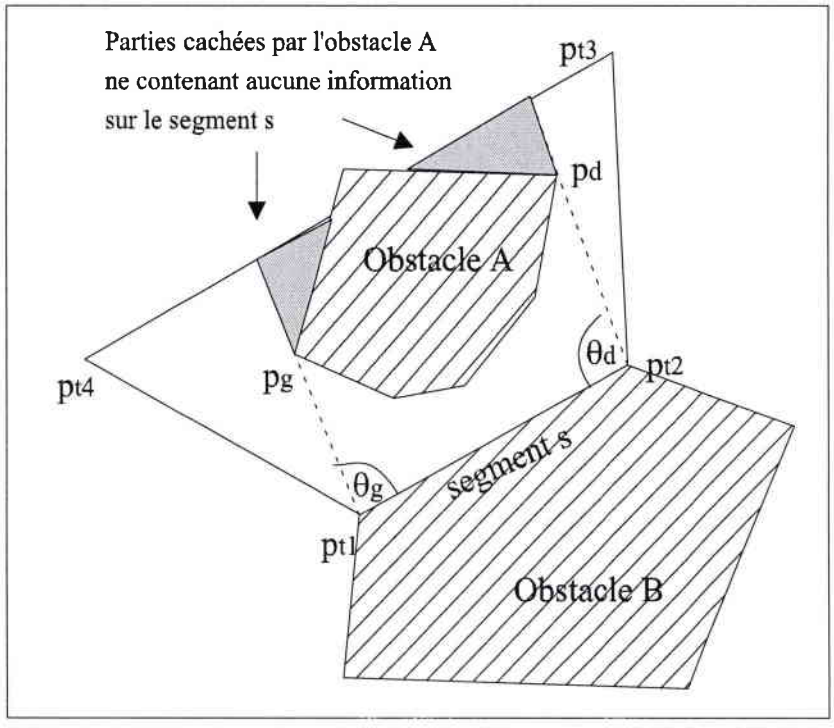


Figure 2.18 : Représentation des angles θ_g et θ_d.

Pour pouvoir trouver les différents cas de parties cachées du trapèze, on doit connaître les valeurs des orientations θ_g et θ_d par rapport à l'angle de réflexion θ_s.

Le calcul de θ_g et θ_d est effectué à partir des produits scalaires des vecteurs constitués respectivement par les points P_g et Pt_1 , Pt_1 et Pt_2, et Pt_2 et P_d.

$$\theta_g = \text{ArcCos}\frac{(P_{gx} - Pt_{1x}).(Pt_{2x} - Pt_{1x}) + (P_{gy} - Pt_{1y}).(Pt_{2y} - Pt_{1y})}{\sqrt{(P_{gx} - P_{t1x})^2.(P_{t2x} - P_{t1x})^2 + (P_{gy} - P_{t1y})^2.(P_{t2y} - P_{t1y})^2}}$$

$$\theta_d = \text{ArcCos}\frac{(P_{dx} - Pt_{2x}).(Pt_{1x} - Pt_{2x}) + (P_{dy} - Pt_{2y}).(Pt_{1y} - Pt_{2y})}{\sqrt{(P_{dx} - P_{t2x})^2.(P_{t1x} - P_{t2x})^2 + (P_{dy} - P_{t2y})^2.(P_{t1y} - P_{t2y})^2}}$$

Pour pouvoir enlever la partie « cachée » du trapèze, il faut déterminer tous les points définissant cette partie.

2.3.3.2 - Calcul des points représentant les parties sans information

On considère trois cas :

■ **1^er cas : si** $\frac{\pi}{2}-\theta_{ref} < \theta_g$ **(et/ou** θ_d**)** $< \frac{\pi}{2}+\theta_{ref}$ **:**

Les coordonnées des deux points à calculer Pi_g et Pi_d des parties cachées du trapèze sont issus des intersections respectivement des deux droites tangentes à l'obstacle aux points P_g et P_d avec le segment haut du trapèze défini par les points Pt_3 et Pt_4 , et passant respectivement par les points Pt_1 et Pt_2 (voir Figure 2.19).

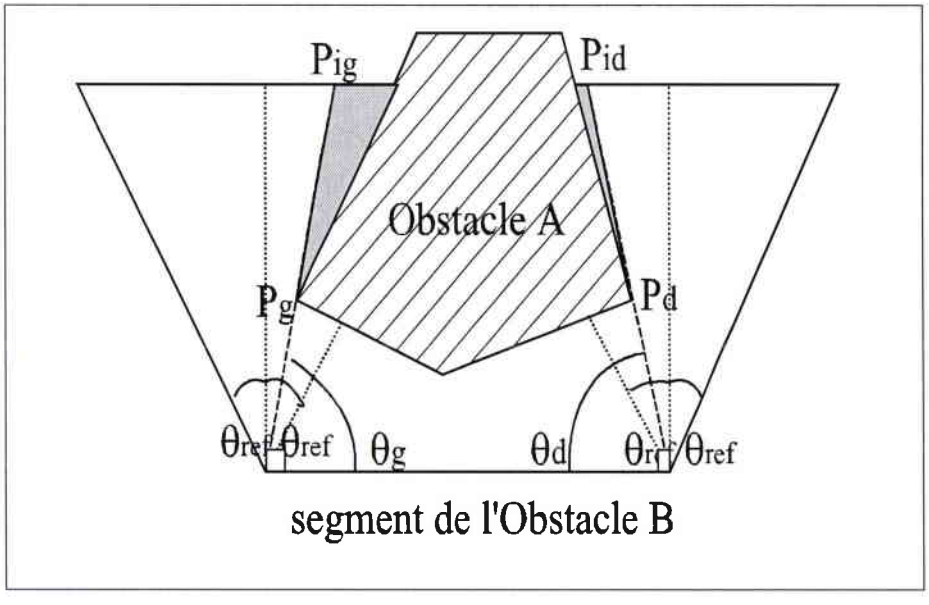


Figure 2.19 : Représentation d'un obstacle traversant un trapèze
dans le cas où $\pi/2-\theta_{ref} < \theta_g$ (et θ_d) $< \pi/2+\theta_{ref}$.

Le point d'intersection p entre deux droites d'équations :

$$y = a_1 x + b_1$$
$$y = a_2 x + b_2$$

est défini en coordonnées cartésiennes (ceci s'applique a Pi_g et Pi_d) par :

$$p_x = \frac{b_1 - b_2}{a_2 - a_1}$$
$$p_y = \frac{a_2.b_1 - a_1.b_2}{a_2 - a_1}$$

■ **2^eme cas : θ_g (et/ou θ_d) < $\pi/2-\theta_{ref}$:**

si $\theta_g < \pi/2-\theta_{ref}$, on calcule les coordonnées du point Pi_g, qui correspond à l'intersection entre la droite parallèle au segment $[Pt_2,Pt_3]$ et passant par le point P_g, et le segment $[Pt_3,Pt_4]$.(fig. 2.20)

si $\theta_d < \pi/2-\theta_{ref}$, on calcule de même les coordonnées du point Pi_d, qui correspond à l'intersection entre la droite parallèle au segment $[Pt_1,Pt_4]$ et passant par le point P_d, et le segment $[Pt_3,Pt_4]$.

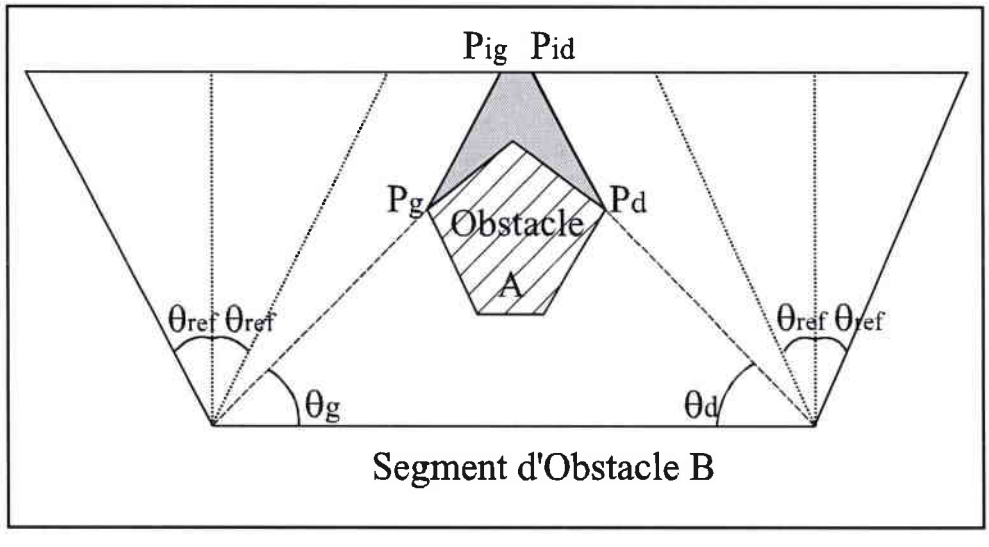


Figure 2.20 : Représentation d'un obstacle traversant un trapèze dans le cas où θ_g (et θ_d) < $\pi/2-\theta_{ref}$.

■ **3eme cas : θ_g (et/ou θ_d) > $\pi/2+\theta_{ref}$**

Si $\theta_g > \pi/2+\theta_{ref}$ alors $P_{ig} = Pt_4$

Si $\theta_d > \pi/2+\theta_{ref}$ alors $P_{id} = Pt_3$

Trois cas possibles peuvent se présenter :

(1) Intersection de l'obstacle (A) avec la limite $[Pt_1,Pt_4]$ du trapèze : P_g correspond au point d'intersection le plus proche de Pt_1 (fig.2.21a).

(2) Intersection de l'obstacle (A) avec la limite $[Pt_2,Pt_3]$ du trapèze : P_d correspond au point d'intersection le plus proche de Pt_2 (fig. 2.21.a).

(3) Intersection de l'obstacle (A) avec la limite $[Pt_3 ,Pt_4]$ du trapèze : P_g correspond au point d'intersection le plus proche de Pt_4, et P_d à celui le plus proche de Pt_3 (fig. 2.21.b).

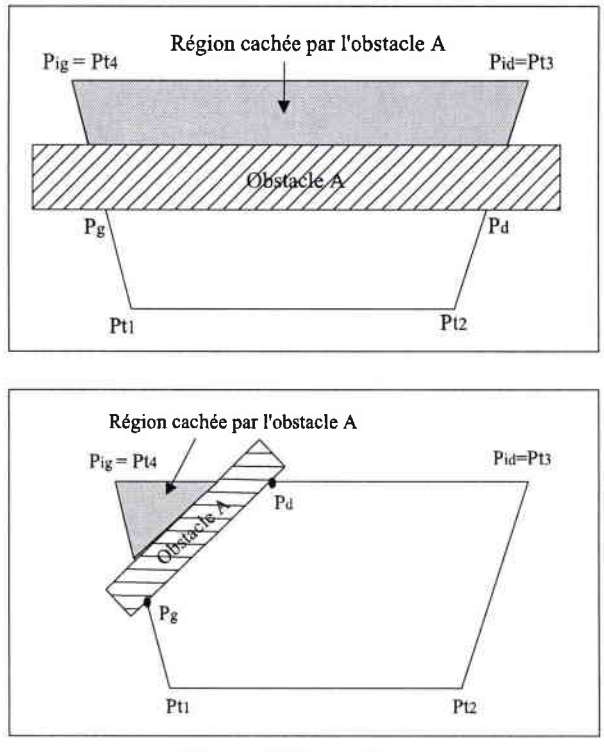


Figure 2.21 a et b

Représentation d'un obstacle traversant deux limites d'un trapèze

2.3.3.3 - Obstruction des parties cachées.

Soit (p_1, p_2, …, p_n) les sommets d'un obstacle qui traverse (ou coupe) un trapèze. Les points sommets sont situés entre les points P_g et P_d dans le sens trigonométrique.

La partie du trapèze à enlever est constituée des points P_{ig}, P_g, P_1, P_2, ...,P_n , P_d, et P_{id} (fig. 2.22a) ou des points P_{ig}, P_g, P_d, et P_{id} (fig. 2.22b)

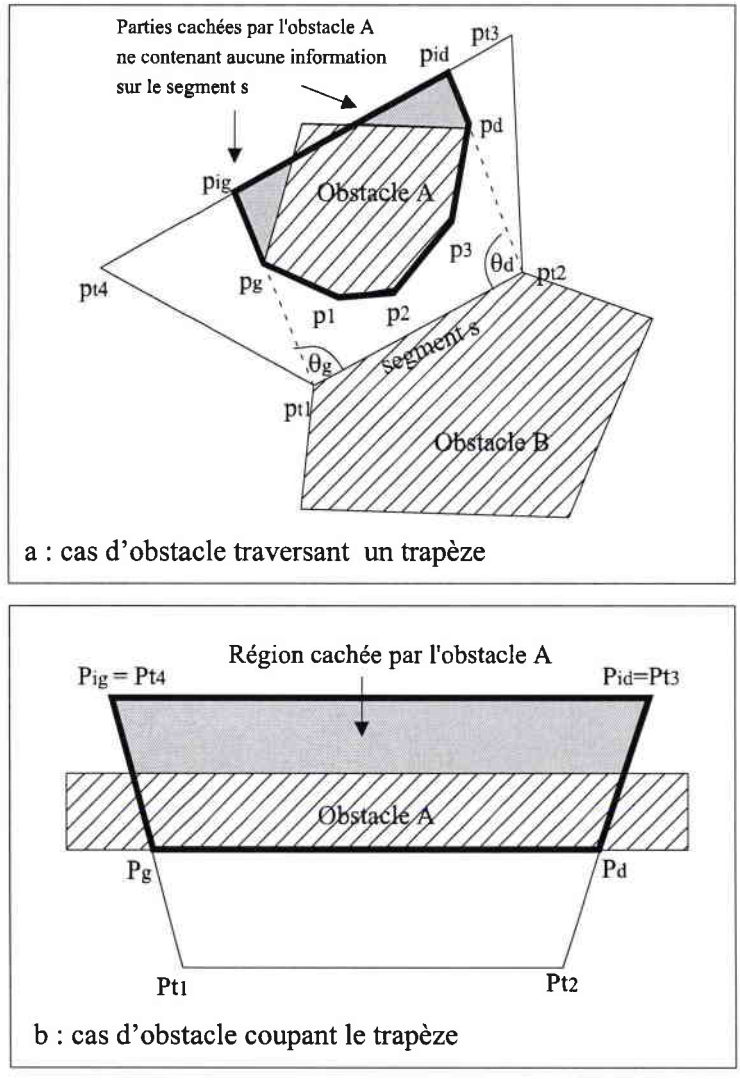


Figure 2.22 a et b

Représentation des parties à enlever
d'un trapèze (en noir foncé)

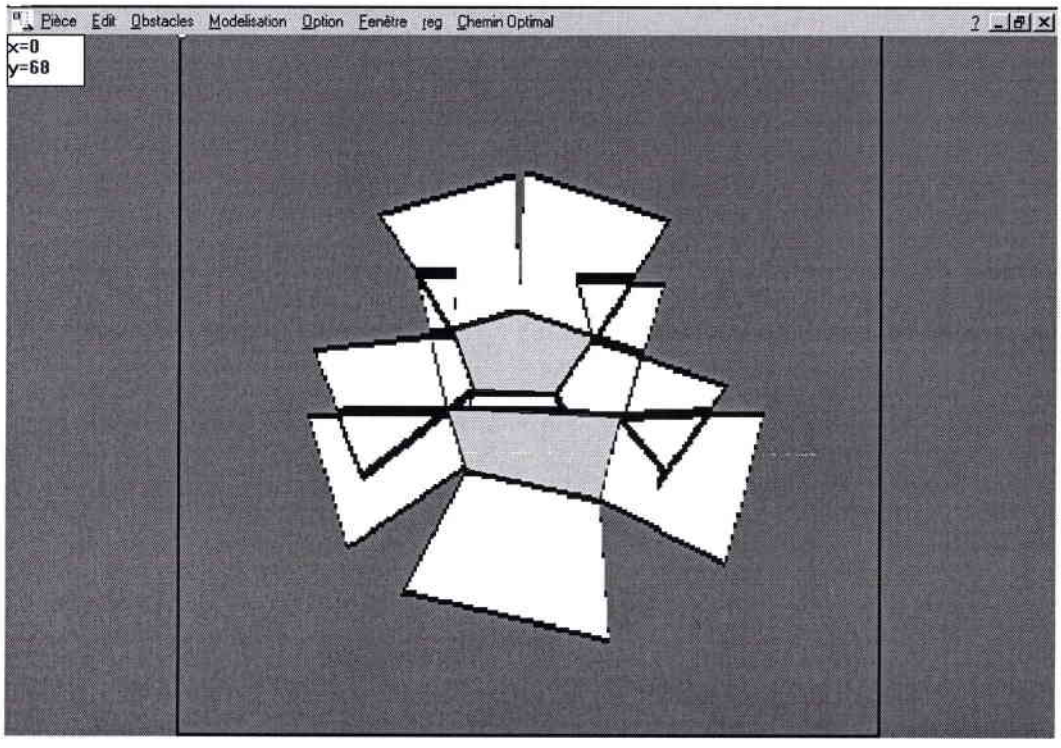


Figure 2.23 (a)

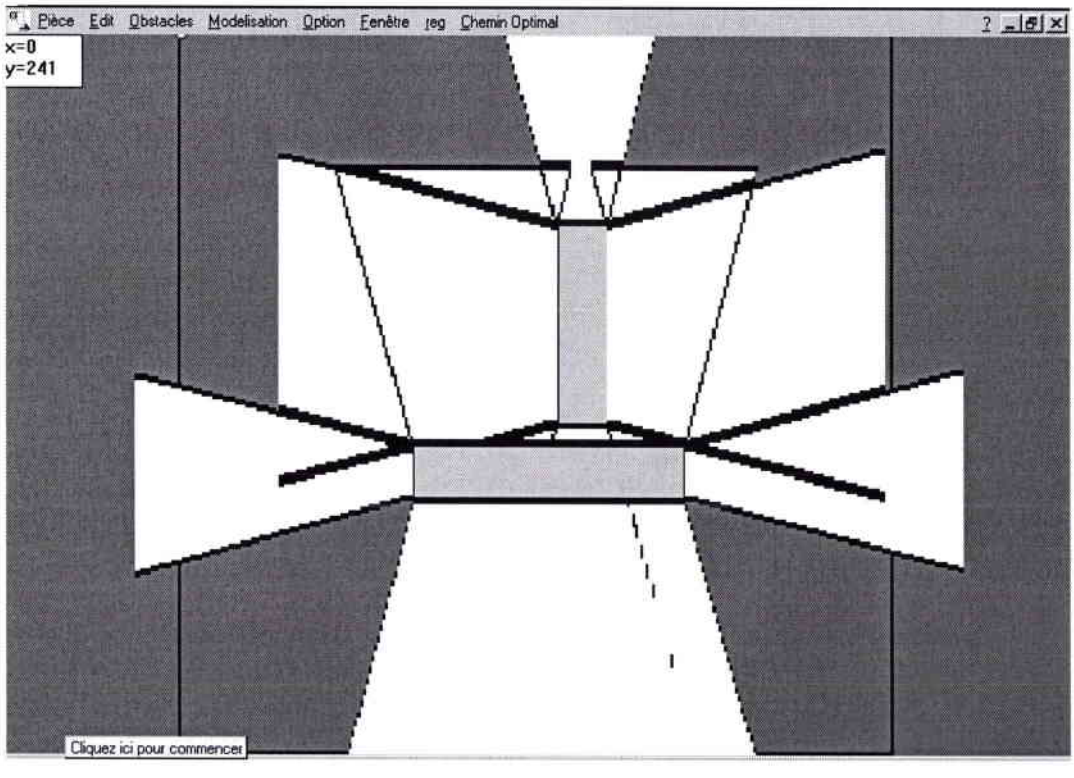


Figure 2.23 (b)

Représentation de deux obstacles ((a) quelconques et (b) rectangulaires avec toutes les régions contenant les informations sur leur segments d'obstacles.

2.4 - Liens entre régions

Lorsque deux trapèzes présentent une intersection, le robot détecte simultanément les deux segments correspondant à ces deux trapèzes, dans la partie qui forme l'intersection. Afin d'obtenir le maximum de robustesse lors de l'application de la commande pour la planification, on considère que le robot ne peut se déplacer d'une région à une autre qu'en changeant un seul état, c. à. d en ajoutant, ou en enlevant, une seule information. Les liens entre les régions vont se faire selon ce changement d'état. Les informations capteurs liées à ces régions trapézoïdales correspondent aux valeurs $\theta°_1$, $\theta°_2$, ..., $\theta°_n$ des signaux $d_1(\theta)$, $d_2(\theta)$, ..., $d_n(\theta)$ définies précédemment.

Divisons l'union de deux trapèzes $T_1 \cup T_2$ ($\cup$ représentant l'opérateur union) en trois parties voir(figure 2.24):

- $T_1 \setminus T_2$ (représente la partie de T_1 sans la partie de T_2).
- $T_1 \cap T_2$ ($\cap$ représentant l'opérateur intersection entre le premier et le deuxième élément).
- $T_2 \setminus T_1$ (T_2 sans T_1).

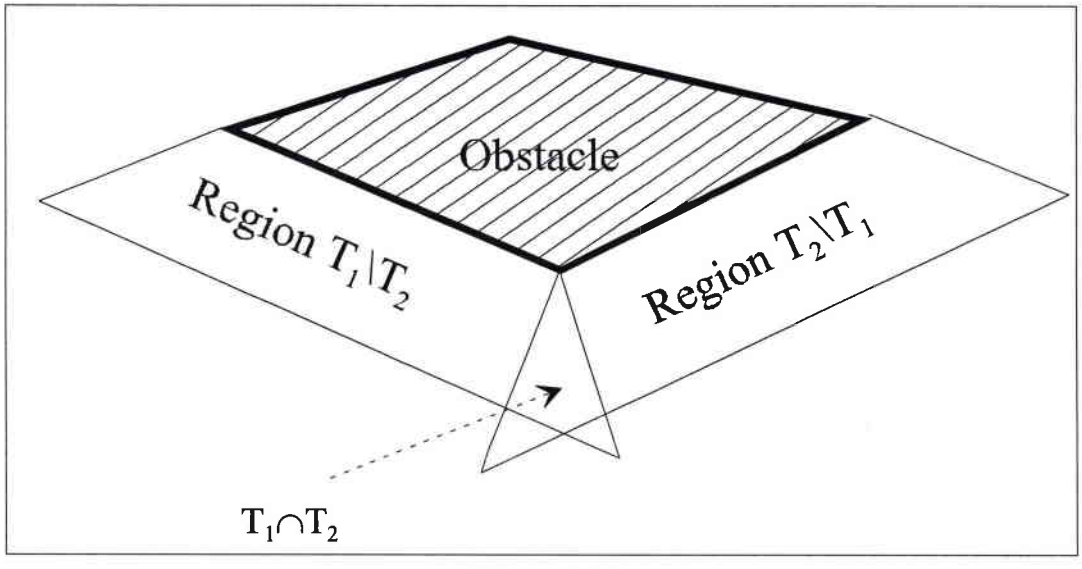


Figure 2.24

Représentation des régions
issues des intersections entre trapèzes

La région $T_1 \backslash T_2$ appartenant au trapèze T_1, le robot détecte la même information que dans T_1, c. à. d. l'angle $\theta°_1$ du sommet du signal $d_1(\theta)$.

De même, $T_2 \backslash T_1$ étant inclus dans la région T_2, le robot va détecter la même information que dans T_2, c. à. d. l'angle $\theta°_2$ du sommet du signal $d_2(\theta)$.

$T_1 \cap T_2$ appartient à la fois à la région T_1 et T_2, et le robot va détecter les deux informations liées respectivement à la région T_1 et T_2, qui sont les angles $\theta°_1$ et $\theta°_2$.

La partie $T_1 \cap T_2$ donnant une information de plus que les deux parties $T_1 \backslash T_2$ et $T_2 \backslash T_1$, il y a donc un changement d'état entre $T_1 \cap T_2$ et les deux autres parties.

Il s'ensuit, que le robot peut directement passer de la région $T_1 \cap T_2$ vers les deux autres régions et vice - versa. La région $T_1 \cap T_2$ va donc servir d'intermédiaire pour le passage du robot de la région $T_1 \backslash T_2$ vers la région $T_2 \backslash T_1$ et inversement.

Si on prend en compte toutes les régions issues des intersections entre les différents trapèzes, il devient incontournable d'utiliser un graphe de noeuds pour établir les régions crées, et leurs liens. Ce sera le sujet du paragraphe suivant.

2.5 - Représentation de graphe pour la modélisation de l'environnement.

2.5.1 - Utilité et application

Les graphes représentent un instrument puissant pour modéliser de nombreux problèmes combinatoires, qui seraient sans cela difficilement abordables par des techniques classiques comme l'analyse mathématique. En plus de son existence en tant qu'objet mathématique, le graphe est aussi une structure de données où les algorithmes, qui permettent de les manipuler, constituent les fondements de l'informatique.

Les graphes sont irremplaçables, dès qu'il s'agit de décrire la structure d'un ensemble complexe en exprimant les relations et dépendances entre ses éléments. Ce mode d'utilisation est très répandu (diagrammes hiérarchiques en sociologie, arbres généalogiques, arbres phylogénétiques en zoologie et botanique, diagrammes de successions de tâches en gestion de projet, etc..)

Un autre mode d'utilisation, très répandu, du graphe est la représentation de connexions et de possibilités de cheminement : détermination de la connexité d'un réseau quelconque (électrique, d'adduction d'eau), calcul du plus court chemin dans un réseau routier, etc..

Les graphes sont enfin précieux pour décrire des systèmes dynamiques, évoluant dans le temps : diagrammes de transition, automates d'états finis, chaînes de Markov.

2.5.2 - Représentation des graphes sur ordinateur

Un graphe de noeuds est constitué d'un ensemble de sommets S et d'arcs A. Soit |A| le nombre d'arcs et |S| le nombre de sommets du graphe.

Il existe deux façons classiques de représenter un graphe G = (S,A) sur ordinateur:

soit par un ensemble de listes d'adjacence, soit par une matrice d'adjacence.

- La représentation par listes d'adjacence est souvent préférée, car elle fournit un moyen peu encombrant de représenter les graphes peu denses, ceux pour qui |A| est très inférieur à $|S^2|$.
- Toutefois, la représentation par matrice d'adjacence sera préférable si le graphe est dense, lorsque |A| est proche de $|A^2|$, ou lorsqu'on veut savoir rapidement s'il existe un arc entre deux sommets donnés.

2.5.1.1 - Représentation par matrice d'adjacence

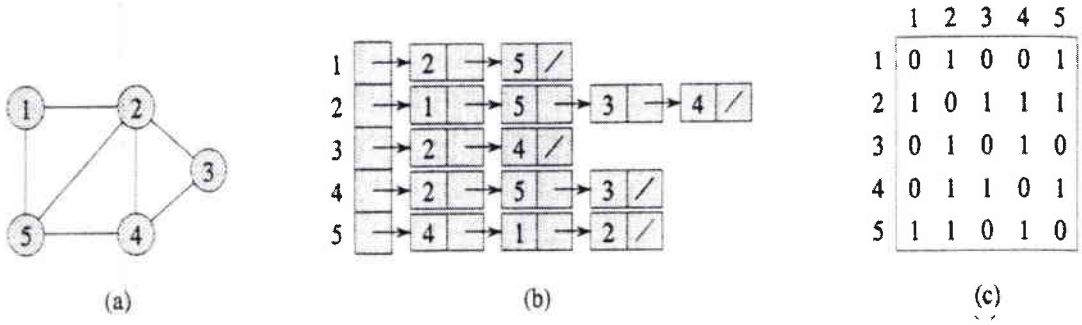
Cette représentation correspond au cas où l'ensemble des sommets du graphe n'évolue pas. On représente l'ensemble des arcs par un tableau . Comme chaque arc est une paire ordonnée de sommets, le graphe est représenté par une matrice carrée de dimensions |S|×|S|, $M = (a_{ij})$, dite ***"matrice d'adjacence"***.

On a :

$$a_{ij} = 1 \text{ si } (i, j) \in A$$

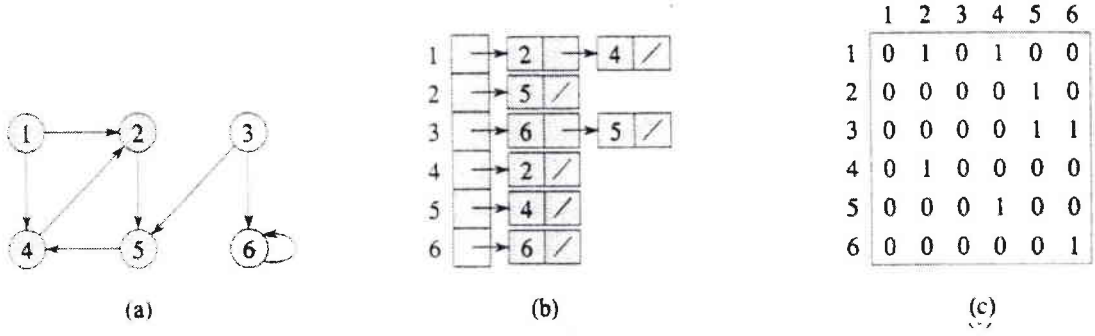
$$0 \text{ sinon}$$

Les figures suivantes sont les exemples de deux graphes (orienté et non orienté), et de leur représentation matricielle, et sous forme de listes.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure 2.25 : Deux représentations d'un graphe non orienté.
(a) Un graphe non orienté G1 possédant cinq sommets et sept arêtes.
(b) Une représentation de G1 par liste d'adjacence.
(c) La représentation de G1 par matrice d'adjacence.



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Figure 2.26 : Deux représentations d'un graphe orienté.
(a) Un graphe orienté G2 possédant six sommets et huit arêtes.
(b) Une représentation de G2 par listes d'adjacence.
(c) La représentation de G2 par matrice d'adjacence.

On a deux conditions :

(1) S'il existe un arc entre les sommets i et j, l'élément d'indices i et j de la matrice M est égal à 1.

(2) S'il n'existe aucun arc entre deux sommets, on peut placer, dans la composante correspondante de la matrice, une valeur autre que 1, qui soit de préférence 0 ou ∞ . (fig. 2.25c et 2.26c)

On définit la transposée d'une matrice $M = (a_{ij})$ comme étant la matrice ${}^tM = ({}^ta_{ij})$ donnée par : ${}^ta_{ij} = a_{ji}$. S'il existe un arc $(u, v) \in A$ dans un arc non orienté, (u, v) et (v, u) représentent alors la même arête, et la matrice d'adjacence M d'un graphe non orienté est donc sa propre transposée : $M = {}^tM$. (figure 2.26c)

Dans certaines applications, il est intéressant de ne conserver que les composants situés sur et au-dessus de la diagonale de la matrice d'adjacence, ce qui réduit presque de moitié la quantité de mémoire requise pour stocker le graphe.

Les matrices d'adjacence peuvent aussi servir à représenter les graphes pondérés. Par exemple, si G = (S, A) est un graphe pondéré associé à une fonction de pondération w (c .à .d. un graphe dont chaque arc possède un poids associé), le poids w (u, v) de l'arc $(u, v) \in A$ est alors simplement stocké à l'intersection de la ligne u et de la colonne v de la matrice d'adjacence.

La représentation matricielle est pratique pour tester l'existence d'un arc (ou d'une arête) entre deux sommets : on accède directement à l'élément de la matrice (en un temps constant). De même, il est facile d'ajouter ou de retirer un arc ou une arête. Il est également facile de parcourir tous les successeurs ou prédécesseurs d'un sommet.

Si le graphe a n sommets, une consultation complète de la matrice requiert un temps d'ordre n^2, et cette représentation exige un espace mémoire de $\Theta(n^2)$, quel que soit le nombre d'arcs ou d'arêtes du graphe. Cela empêche d'avoir des algorithmes d'ordre inférieur à n^2 pour des graphes de n sommets, n'ayant que peu d'arcs. Pour remédier à cet inconvénient, on utilise, dans ce cas, la représentation par listes d'adjacence.

2.5.2.2 - Représentation par listes d'adjacence.

La représentation par listes d'adjacence d'un graphe **G = (S, A)** consiste en un tableau Adj de $|S|$ listes, une pour chaque sommet de l'ensemble S. Pour chaque $\mathbf{u} \in \mathbf{S}$, la liste d'adjacence Adj[u] est une liste chaînée des sommets v, tels qu'il existe un **arc (u, v)** $\in$ **A**. Autrement dit, Adj[u] est constituée de tous les sommets adjacents à **u** dans **G**. Les sommets de chaque liste d'adjacence sont en général chaînées selon un ordre arbitraire.(fig. 2.25b et 2.26b).

Selon le type de graphe utilisé, on a les conditions suivantes :

- Si G est un graphe orienté, la somme des longueurs de toutes les listes d'adjacence vaut $|A|$, puisque l'existence d'un arc de forme (u, v) se traduit par la présence de v dans Adj(u).
- Si G est un graphe non orienté, la somme des longueurs de toutes les listes d'adjacence vaut $2\times|A|$, puisque si (u, v) est une arête, u apparaît dans la liste d'adjacence de v, et vice versa.

Qu'un graphe soit orienté ou non, la représentation par listes d'adjacences possède la propriété avantageuse de ne demander qu'une quantité de mémoire limitée à **O(max (S, A)) = O(S+ A).**

Suivant les cas et la nature des problèmes, on constituera des listes de sommets, ou des listes d'arcs, en précisant au besoin certaines des informations qui peuvent leur être attachées (comme le degré des sommets, la longueur des arcs, le fait qu'un sommet a été déjà visite, etc)

Les listes d'adjacence peuvent aisément être adaptées aux graphes pondérés, représentés habituellement par une fonction de pondération **w : A → R**. Le poids **w (u, v)** de l'arc **(u, v)** ∈ **A** est simplement stocké avec le sommet **v** dans la liste d'adjacence de **u**.

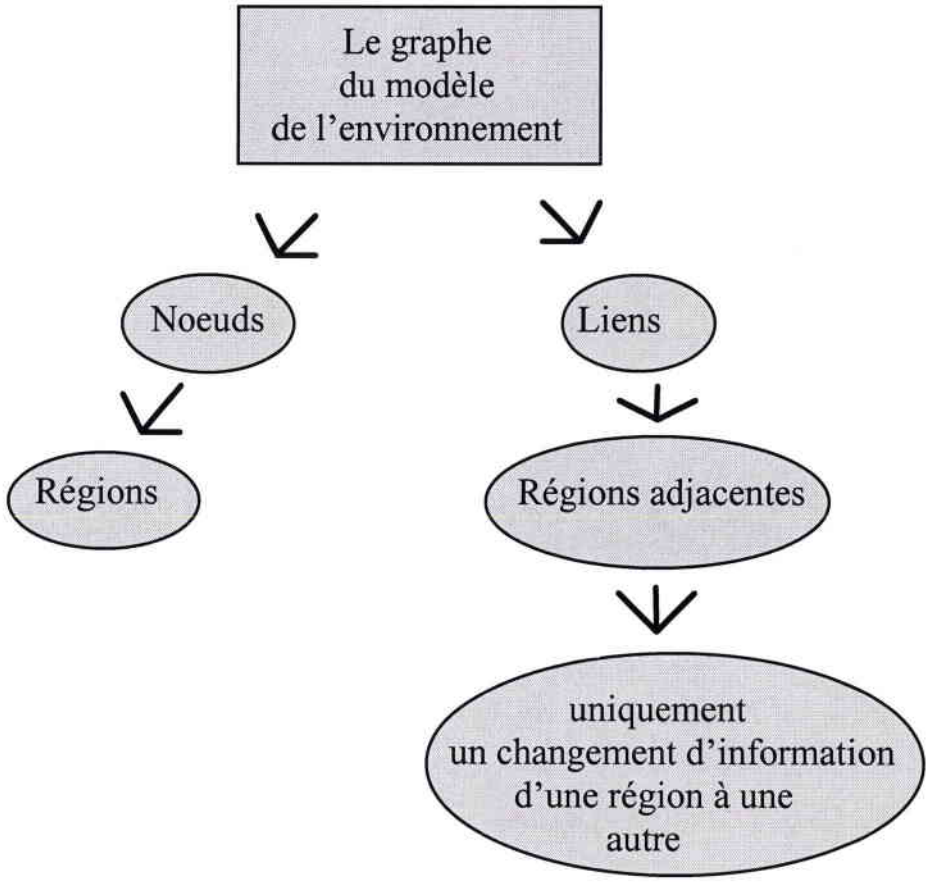
La représentation par listes d'adjacence est assez robuste, dans le sens où elle peut être modifiée pour supporter de nombreuses variantes sur le graphe. C'est aussi la représentation la mieux adaptée aux structures évolutives, car elle permet de ne parcourir que les éléments d'un ensemble de base E_b ,E_b étant lui même un sous-ensemble du graphe.

Les inconvénients de ce genre de représentations viennent essentiellement:

- du fait qu'il est nécessaire de programmer les opérations de gestion de listes habituelles: création, adjonction, suppression d'objets, parcours et recherche.
- et aussi du fait que les opérations et prédicats ensemblistes ne sont pas pré-programmés, et qu'il faut les construire en fonction des problèmes à traiter.

Pour notre part, on a choisi la représentation par listes d'adjacence, parce qu'elle demande moins d'espace mémoire. La construction du graphe est réalisée à partir d'un algorithme qui utilise une liste de noeuds (structures) chaînés. Les noeuds du graphe vont représenter toutes les régions crées à partir des informations capteurs.

La construction de ce graphe de noeuds est réalisée à partir des liens entre ces régions. Le lien entre deux régions est établi s'il y a un changement d'état entre ces deux régions (voir graphe ci dessous).



La construction du graphe d'une part sert à trouver tous les chemins possibles entre les régions et d'autre part sert d'étape pour optimiser la trajectoire du robot.

2.6 - Complexité de l'algorithme pour la construction du graphe

Avant de définir l'algorithme pour la construction du graphe, il serait intéressant de déterminer la complexité en nombre de régions, c. à. d. le nombre maximum de régions créées en fonction du nombre **n** de segments d'obstacles (ou du nombre de trapèzes).

La complexité Rc_n est en fait une somme de combinaisons, définie par :

$$Rc_n = C_n^1 + C_n^2 + \ldots + C_n^j + \ldots + C_n^n$$

avec

$$C_n^j = \frac{n!}{(n-j)! \times j!}$$

! étant l'opérateur factoriel, **n!** est défini par :

$$n\,! = n \times (n-1) \times (n-2) \times \ldots (n-j) \times \ldots \times 2 \times 1$$

C^1_n représente le nombre de régions qui n'appartiennent qu'à un seul trapèze ou qui ne détectent qu'un seul segment. Ce sont donc les régions qui ont une seule information.

C^j_n représente le nombre de régions qui sont issues de l'intersection entre **j** trapèzes ou qui appartiennent à **j** trapèzes. Elles ont donc **j** informations.

C^n_n représente une seule région qui est issue de l'intersection entre tous les trapèzes. Elle appartient à tous les trapèzes et à **n** informations.

On désigne les **n** trapèzes par (T_1, T_2, …, T_n), et on vérifie le nombre d'intersections entre trapèzes :

- S'il y a une intersection entre T_1 et T_2, la région T_1 est divisée en deux parties : $T_1 \setminus T_2$ et $T_1 \cap T_2$. En plus de ces deux parties, on a la partie de T_2 qui n'appartient pas à T_1 ($T_2 \setminus T_1$), on obtient donc : **2*1 (2 parties de T_1) + 1 ($T_2 \setminus T_1$) = 3 régions**
- si une région T_3 représente une intersection respectivement avec ces trois régions, celles ci seront à leur tour divisées en deux. En plus, une septième région sera constituée de la

partie de T_3 qui n'appartient pas aux trapèzes T_1 et T_2 . On aura donc : **2* 3 (tous les points de $T_1 \cup T_2$) + 1 ($T_3 \setminus T_1 \cup T_2$) = 7 régions**

On constate que à chaque fois qu'un trapèze présente une intersection avec des sous-régions, le nombre de nouvelles régions est augmenté d'un facteur 2 plus 1. Si on applique ce raisonnement sur n trapèzes, on aura:

$$Rc_n = Rc_{n-1}\times 2 + 1$$
$$= (Rc_{n-2}\times 2 + 1)\times 2 + 1 = Rc_{n-2}\times 2^2 + Rc_{n-2}\times 2^1 + 1$$
$$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$= Rc_{n-j}\times 2^j + Rc_{n-j}\times 2^{j-1} + \ldots + Rc_{n-j}\times 2^1 + 1$$
$$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$= Rc_1\times 2^{n-1} + Rc_1\times 2^{n-2} + \ldots + Rc_1\times 2^1 + 1 \ (Rc_1=1)$$
$$= 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 1$$

Comme on obtient une suite géométrique pour un entier $x \neq 1$, la sommation:

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n$$

a pour valeur :

$$\sum_{k=0}^{n} x^k = \frac{(x^{n+1} - 1)}{x - 1}$$

D'ou la fonction permettant de déterminer le nombre de combinaisons:

$$\boxed{Rc_n = 2^n - 1}$$

On peut vérifier cette valeur Rc_n en utilisant les éléments binaires 0 et 1, selon les informations captées sur les segments d'obstacles: comme on a n segments, le nombre maximum de combinaisons devrait être 2^n. Cependant, en éliminant le cas où tous les éléments linéaires sont égaux à zéro (c. à. d le cas où il n'y a aucune information sur aucun

segment), le nombre max. de combinaisons Rc_n redevient 2^n-1, c. à. d le même que la formule précédente.

Prenons deux exemples :

- supposons avoir 10 segments (n=10), et que les régions contenant l'information sur les segments sont d'indices 2, 5 et 7. La valeur binaire sera alors:

0001010010

la même valeur Rc_n en nombres décimaux sera:

$2^{2-1} + 2^{5-1} + 2^{7-1} = 2 + 16 + 64 = 82$ combinaisons possibles

- supposons avoir 10 segments, et que les régions contiennent des informations sur tous les segments, c. à. d. la valeur binaire maximum:

1111111111

la valeur Rc_n en nombres décimaux sera alors :

$2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9 =$

$2^{10} - 1 = 1023$ combinaisons.

La complexité augmente rapidement (de façon exponentielle), en fonction du nombre de segments.

En pratique, il en va autrement: la probabilité qu'une région appartienne à **n** trapèzes diminue lorsque **n** augmente. Si une région appartient à **n** trapèzes, on procède à **n** divisions pour avoir cette région. Ce qui fait qu'une région qui appartient à un nombre assez élevé de trapèzes sera suffisamment petite pour être négligeable.

Il est rare en pratique de voir une région appartenir à plus de trois trapèzes. La complexité est donc beaucoup moins élevée qu'en théorie.

2.7 - Algorithme pour la construction du graphe

Après avoir abordé la complexité des régions, on peut envisager la construction du graphe. Rappelons que le graphe à construire est constitué d'un ensemble de sommets et d'arcs. Les sommets sont reliés entre eux par les arcs. Les sommets représentent les régions, et les arcs les liens qui existent entre les régions.

Un lien existe entre deux régions lorsque le robot passe directement d'une région a une autre. Les liens entre deux régions sont réciproques: si on passe de la région T_1 à T_2, on peut aussi passer de T_2 à T_1. Les arcs (et le graphe) ne sont donc pas orientés. Il n'y a pas de différence entre l'origine et l'extrémité d'un arc, puisque les deux sommets qui sont liés par cet arc pointent l'un vers l'autre. Le couple constitué des deux sommets forme une boucle.

2.7.1 - Types de liens entre noeuds rencontrés

Avant d'établir l'algorithme de construction du graphe, il est nécessaire de déterminer tous les types de liens entre noeuds:

Soit **n** le nombre de segments d'obstacles, donc le nombre de régions (trapèzes) associées a chacun des segments. Les noeuds représentent les sous-régions crées à partir des opérations (intersection $\cap$ et exclusivité /) sur les trapèzes T_1 à T_n .

Pour indiquer la variation d'information qui s'effectue en passant d'un noeud à l'autre, Il faut déterminer, pour chaque noeud, l'information sur tout segment détecté dans la sous-régions représentée par ce noeud. On utilise pour cela le code binaire (0, 1). Les segments sont numérotés $S_1, S_2,\ldots,S_n$.

Tableau 1 : (n =2)

Sous-régions	Segments	Liens	
	S_2 S_1	1	2
$T_1 \backslash T_2$	0 1	↕	
$T_1 \cap T_2$	1 1		↕
$T_2 \backslash T_1$	1 0		

Tableau 2 : (n =3)

Sous-régions	Segments	Liens entre sous-régions						
	S_3 S_2 S_1	1	2	3	4	5	6	7
$(T_1 \setminus T_2) \setminus T_3$	0 0 1	↕		↕				
$(T_1 \cap T_2) \setminus T_3$	0 1 1	↓	↕		↕			
$(T_2 \setminus T_1) \setminus T_3$	0 1 0		↓			↕		
$(T_1 \setminus T_2) \cap T_3$	1 0 1	↕		↓			↕	
$(T_1 \cap T_2) \cap T_3$	1 1 1	↓	↕		↓			
$(T_2 \setminus T_1) \cap T_3$	1 1 0		↓			↓		↕
$T_3 \setminus (T_1 \cup T_2)$	1 0 0						↓	↕

Tableau 3 : (n = 4)

Sous-régions	Segments	Liens entre sous-régions																	
	S_4 S_3 S_2 S_1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
$((T_1 \setminus T_2) \setminus T_3) \setminus T_4$	0 0 0 1	↕		↕					↕										1
$((T_1 \cap T_2) \setminus T_3) \setminus T_4$	0 0 1 1	↓	↕		↕					↕									2
$((T_2 \setminus T_1) \setminus T_3) \setminus T_4$	0 0 1 0		↓			↕					↕								3
$((T_1 \setminus T_2) \cap T_3) \setminus T_4$	0 1 0 1	↕		↓			↕					↕							4
$((T_1 \cap T_2) \cap T_3) \setminus T_4$	0 1 1 1	↓	↕		↓								↕						5
$((T_2 \setminus T_1) \cap T_3) \setminus T_4$	0 1 1 0		↓			↓		↕						↕					6
$(T_3 \setminus (T_1 \cup T_2)) \setminus T_4$	0 1 0 0						↓	↕							↕				7
$((T_1 \setminus T_2) \setminus T_3) \cap T_4$	1 0 0 1	↕		↕					↓							↕			8
$((T_1 \cap T_2) \setminus T_3) \cap T_4$	1 0 1 1	↓	↕		↕					↓							↕		9
$((T_2 \setminus T_1) \setminus T_3) \cap T_4$	1 0 1 0		↓			↕					↓							↕	10
$((T_1 \setminus T_2) \cap T_3) \cap T_4$	1 1 0 1	↕		↓			↕					↓							11
$((T_1 \cap T_2) \cap T_3) \cap T_4$	1 1 1 1	↓	↕		↓								↓						12
$((T_2 \setminus T_1) \cap T_3) \cap T_4$	1 1 1 0		↓			↓		↕						↓					13
$(T_3 \setminus (T_1 \cup T_2) \cap T_4$	1 1 0 0						↓	↕							↓				14
$T_4 \setminus (T_1 \cup T_2 \cup T_3)$	1 0 0 0															↓	↓	↕	15

Trois catégories de liens apparaissent dans ces tableaux:

- La 1^{ere} catégorie est celle des liens dérivés du tableau 1 et repris dans le tableau 2 pour être associés au nouveau élément du 2eme tableau (ainsi tous les liens inclus dans les colonnes verticales 1 et 2 du 2eme tableau dérivent du 1er tableau, tout en formant de nouvelles combinaisons avec le nouveau élément T_3). La 1^{ere} catégorie est transmise, de la même façon, au tableau 3 à partir du tableau 2 (tous les liens inclus dans les colonnes verticales 1 à 7 du tableau 3).
- La 2eme catégorie est celle des liens qui, au niveau du 2eme tableau, sont crées entre les sous-régions complémentaires par rapport à T_3 (c. à. d. entre les sous-régions qui constituent le complément des sous-régions du tableau 1 par rapport à T_3 et celles qui constituent les intersections avec T_3 (colonnes verticales 3 à 5). La 2eme catégorie est, de même, transmise au tableau 3 (colonnes 8 à 14).
- La 3eme catégorie est celle des liens de la partie du nouveau élément de chaque tableau, (T_3 pour tableau 2, T_4 pour tableau 3) qui n'a aucune intersection avec les sous-régions des tableaux précédents.

Chacune de ces trois catégories de liens sera traitée séparément dans l'algorithme de construction du graphe.

2.7.2 - Programmation dynamique

Pour pouvoir traiter un grand nombre d'éléments (jusqu'à 2^n-1), on utilise la programmation dynamique introduit par les travaux de Richard Bellman [Bel 57], qui permet d'atténuer le caractère combinatoire de certains problèmes et rendant ainsi matériellement possible la détermination d'un optimum.

La programmation dynamique est une méthode de résolution des problèmes d'optimisation basé sur une énumération explicite des solutions. Le mot programmation ne doit pas être compris dans le sens qu'on lui donne en informatique, il signifie précisément résolution de problème. Le mot dynamique signifie que le temps intervient d'une façon cruciale dans la résolution. La programmation dynamique exige que les problèmes traités aient une structure particulière, de type séquentiel. Ce qui veut dire que le problème est décomposé en plusieurs parties.

Les algorithmes utilisant la programmation dynamique partitionnent le problème en sous-problèmes, qu'ils résolvent récursivement, puis combinent leurs solutions pour résoudre le problème initial. Le développement d'un tel algorithme peut être planifié en trois étapes :

1. Caractériser la structure d'une solution optimale.
2. définir récursivement la valeur d'une solution optimale
3. Calculer la valeur d'une solution optimale, en remontant progressivement jusqu'à l'énoncé du problème initial.

Le lecteur désireux d'approfondir le sujet pourra se référer au livre de J. L. Laurière [Lau 79].

2.7.3 - Principe de l'algorithme pour la construction du graphe

- Soit **Liste_Trapèzes :** la liste représentant les trapèzes T_1 à T_n associés aux segments d'obstacles
- et **I** le nombre de régions en cours de traitement. On initialise les liens pour $T_1 \setminus T_2$ et $T_1 \cap T_2$

L'algorithme consiste à construire les régions en trois parties:

- La première partie consiste à mettre dans Liste1 la différence entre le I^{eme} et le $I\text{-}1^{eme}$ éléments de **Liste_Trapèzes** : $T_I \setminus T_{I-1}$. On garde les mêmes liens que pour le $I\text{-}1^{eme}$ traitement.
- La deuxième partie consiste à mettre dans Liste2 l'intersection entre le I^{eme} et le $I\text{-}1^{eme}$ éléments de **Liste_Trapèzes** : $T_I \cap T_{I-1}$. On garde les mêmes liens que pour le $I\text{-}1^{eme}$ traitement.
- La troisième partie consiste à calculer un élément "d'exclusion" due au fait qu'il est calculé en dehors des deux listes précédentes. Cette partie consiste à prendre le I élément de **Liste_Trapèzes** sans les éléments précédentes, c. à. d $T_I \setminus (T_{I-1} \setminus (T_{I-2} \setminus (\ldots \setminus T_1)))$.

Après chaque traitement (lorsque I augmente de 1), la deuxième liste est liée à la première, et l'élément d'exclusion est inséré à la fin de la deuxième liste.

Les noeuds de **Liste1** sont numérotés de 1 a 2^{I-1}-1, ceux de **Liste2** sont numérotés de 2^{I-1} à 2^I-1, et l'élément d'exclusion à 2^I-1.

Pour chaque traitement, il faut établir les liens suivants:

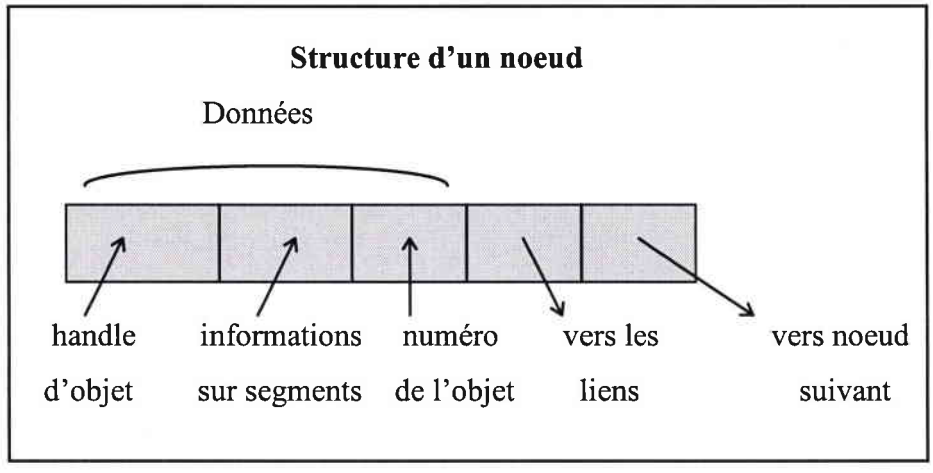
- entre **Liste1** et **Liste2**: de j=1 à l'élément 2^{I-1}-1, lien entre élément j et j+2^{I-1}-1.
- entre **Liste2** et l'élément d'exclusion X situé à 2^I-1, pour j=1 à I-2, lien entre élément X et (2^I-1)+ 2^0, X et (2^I-1)+ 2^0 + 2^1, … et X et (2^I-1)+ 2^0 + 2^1 + … + 2^j.

On fait le calcul jusqu'au dernier élément de **Liste_Trapèzes**.

Les régions qui ont plus d'une information ont au maximum n-1 liens, ceux qui ont plus d'informations ont au maximum n liens.

2.7.4 - Représentation informatique

Comme précédemment mentionné, on utilise des listes de noeuds pour la représentation informatique du graphe. Chaque noeud est une structure qui contient plusieurs membres. (schéma ci-dessous).



Pour pouvoir passer d'un noeud au suivant, on donne à la structure de chaque noeud les caractéristiques d'une structure auto-référentielle, c. à. d une structure dont un des membres est un pointeur orienté vers l'adresse du noeud suivant dans la même liste, et dont d'autres membres pourraient aussi être des pointeurs orientés vers d'autres noeuds avec lesquels notre noeud a des liens (tableau [n] de liens mentionné ci dessous).

La structure du noeud aura ainsi la forme suivante :

```
Structure noeud   {
                      vers noeud suivant
                      tableau[n] de liens vers des noeuds
                      identificateur de l'objet région représentant le noeud (handle)
                      numéro affecté au noeud
                      liste des informations sur les segments d'origine
                  }
```

En informatique, une région est considérée comme un objet. Un objet est une structure interne qui représente une ressource système, comme un fichier, une image graphique, une fenêtre, une région de l'environnement dans notre cas.

Une application ne peut pas directement accéder à la structure interne d'un objet ou du ressource système qu'un objet représente. A la place, une application doit utiliser un « handle » d'objet et s'en servir pour examiner et modifier le contenu de l'objet ou du ressource système. A chaque fois qu'on veut appliquer des opérations sur des régions comme par exemple l'union, l'intersection ou l'obstruction, on fait appel au handle de cette région qui permet de l'identifier.

En C++, on utilise le symbole → pour pointer à un membre d'une structure noeud, mais aussi vers une autre structure noeud avec le membre suivant et avec tableau[n] de liens.

Dans les définitions qui vont suivre, nous utilisons les notations logiques du langage C++ indiquées dans le tableau de la figure 2.27.

Symbole	Opération logique
&&	ET Booléen entre deux propositions
||	OU Booléen entre deux propositions
==	Comparaison
!=	Différence

Figure 2.27 : Notation

2.8 - Implantation des procédures pour la construction du graphe

Une procédure informatique est une fonction qui permet de réaliser des opérations basées sur un ensemble d'informations reçues. Nous allons, pour la construction de notre graphe, utiliser :

(1) une procédure pour préparer la "copie d'une liste"
(2) deux procédures pour réaliser les opérations arithmétiques sur les régions
(3) trois procédures pour établir les liens entre noeud, quand ils existent.
(4) une procédure pour insérer une liste derrière une autre.
(5) une procédure pour enlever un ou plusieurs noeuds du graphe.

2.8.1- Copie d'une liste

Dans l'exercice de recherche de chemins, il est utile d'avoir, sur une liste séparée, toutes les informations sur les régions issues des intersections de trapèzes, car, comme on l'a déjà vu, ce sont les liens entre ces intersections qui créent des chemins.

L'utilité d'avoir deux listes dont les informations sont organisées comme indiqué ci-dessous contribue à faciliter l'établissement des liens entre régions :

- **i** étant le nombre de trapèzes dont l'intersection, à un instant donné, a introduit un certain nombre de sous régions,
- **Liste1** étant la liste de tous les éléments (noeuds) représentant les sous-régions crées à la **i-1**^{ème} itération.
- et **Liste2** étant une liste initialement vide, qui se remplit graduellement de tous les éléments de **Liste1** ayant des intersections avec le nouveau trapèze T_i , à la i^{eme} itération .

La copie de **Liste1** consiste à créer une nouvelle liste d'adresse **Liste2**, tel que :

- le contenu de **Liste2** (c. à. d. les données) soit identique à celui de **Liste1**.
- l'adresse de **Liste2** soit différente de celle de **Liste1**.

Dans notre cas, il s'agit d'une copie partielle, sous condition.

Dans cette partie, on va essayer de copier, dans la **Liste2**, tous les éléments de **Liste1** représentant les régions crées à la i-1ème itération, qui ont des intersections avec le nouveau trapèze de la ième itération T_i. **Liste 2** va contenir par la suite les noeuds représentant toutes ces intersections.

La condition à la ième itération qui permet de copier le contenu d'un noeud de Liste1 dans celui de Liste2 est la suivante :

> **Si** l'intersection est vrai entre une région d'un noeud de liste
> avec le nouveau trapèze T_i
> **Alors** Copie du contenu du noeud de Liste1 dans celui de Liste2

Exemple :

Soit $R_1 = T_1 \backslash T_2$, $R_2 = T_1 \cap T_2$ et $R_3 = T_2 \backslash T_1$ les régions représentés par les noeuds successives d'une liste . R_1 et R_2 présentent une intersection avec le trapèze T_3 associé à un segment d'obstacle, mais pas R_3. Liste 2 va contenir uniquement les noeuds représentant R_1 et R_2.

La procédure qui permet de copier une liste sous condition à la ième itération est la suivante :

```
1   CopieListe(liste : pointeur ;listecopie : pointeur)
2   {
3       Tant que liste ≠ Nil alors
4           Si Intersection(liste→Région, Ti) == vrai alors
5               Liste2→Région = Liste1→Région ;
6               Liste2 = Liste2 → suivant ;
7               Liste1 = Liste1 → suivant ;
8           Liste2 = Nil ;
    }
```

La procédure **CopieListe** fonctionne de la manière suivante :

- Ligne 3-7 → Tant que **Liste1**, qu'on veut copier n'est pas nulle (vide), on vérifie s'il y a intersection entre la région pointée par le noeud de **Liste1** et le nouveau trapèze. Si c'est le cas, alors l'élément actuel de **Liste2** reçoit l'information de l'élément (donnée sur une région) actuel de **Liste1**. Ensuite le pointeur de **Liste2** passe de l'élément actuel à l'élément suivant de **Liste2**. Le pointeur de **Liste1** passe aussi à son élément suivant. On parcours ainsi tout **Liste1**, et à chaque fois qu'il y a intersection entre les nouvelles régions créées et un nouveau trapèze T_i, on copie le contenu de l'élément de **Liste1** dans celui de **Liste2**.
- Ligne 8 → Lorsque le pointeur de **Liste 1** est parvenue au bout de la liste, le pointeur **Liste2** aura atteint lui aussi la fin de la liste.

Liste 2 est une copie partielle de **Liste1**. L'établissement de ces deux listes permet d'utiliser plus loin les opérations arithmétiques de complément (\) et d'intersection ($\cap$) en fonction du nouveau trapèze introduit respectivement sur les régions représentées par **Liste1** et **Liste2.**

2.8.2 - Opérations sur les régions

Pour réunir, à une itération **i** donnée, les informations sur les sous-régions issues des intersections (entre les sous régions issues de l'itération précédente, et le nouveau trapèze A_i introduit avec l'itération **i**), on utilise les opérations de " complément d'une région par rapport à une autre", et "d'intersection entre deux régions " .

2.8.2.1 - Complément d'une région par rapport à une autre

Rappelons que le complément d'une région A_2 par rapport à une autre région A_1 est $A_1 \backslash A_2$:

- si A_2 et A_1 présente une intersection, le complément de A_2 est la partie de A_1 qui est en dehors de l'intersection
- si A_1 et A_2 ne présente aucune intersection, alors le complément de A_2 est A_1 tout entier.

A une certaine itération, on remplace les informations sur les sous-régions de l'itération précédente, représentées par les noeuds de **Liste1**, par les informations constituant les compléments de ces sous-régions par rapport au nouveau trapèze A_2 introduit.

Si A_1 est une sous-région représentée par un noeud de **Liste1**, et A_2 le nouveau trapèze, on effectue, pour chaque noeud de **Liste1**, l'application suivante :

$$A_1 \leftarrow A_1/A_2$$

La procédure qui permet de réaliser cette opération est définie par :

```
Graphe_Complément(pointeur : Liste1, Région : nouveau_trapèze)
{
   si( Liste1 != NULL)
      Liste1→Région = Complément(Liste1→Région, nouveau_trapèze);
      Graphe_Complément(Liste1→suivant, nouveau_trapèze);
}
```

2.8.2.2 - Intersection entre deux régions

A une certaine itération, on remplace les informations sur les sous-régions de l'itération précédente, représentées par noeuds de **Liste2**, par les informations sur les sous-régions constituant les intersections de ces sous-régions par rapport au nouveau trapèze A_2 introduit.

Si A_1 représente la région d'un noeud de **Liste2** et A_2 un nouveau trapèze, il faut faire l'application suivante pour chaque noeud de **Liste2** :

$$A_1 \leftarrow A_1 \cap A_2$$

La procédure qui permet de réaliser cette opération est définie par :

```
Graphe_Insertion(pointeur : Liste2, Région : nouveau_trapèze)
{
    Si( Liste2 != NULL)
        Liste2→Région = Intersection(Liste2→Région, nouveau_trapèze);
        Graphe_Insertion(Liste2→suivant, nouveau_trapèze);
}
```

2.8.3 - Liens entre les noeuds du graphe

On a vu au chapitre 3.3.1, qu'on avait trois catégories de liens entre sous-régions. On en avait fait l'illustration en utilisant des tableaux de sous-régions. Chaque catégorie de liens sera traitée séparément dans l'algorithme de liaison entre les noeuds du graphe, en utilisant pour chacune une procédure différente:

- La procédure qui permet d'établir la première catégorie de liens consiste à chercher les liens qui existent, à une itération donnée, entre les noeuds de **Liste1** et ceux de **Listes2**. Elle correspond donc à la liaison entre listes (**Liste1** et **Liste2**).
- La procédure qui permet d'établir la deuxième catégorie de liens consiste à chercher les liens qui existent, à une itération donnée, entre tous les noeuds de **Liste2**. Elle correspond donc à la liaison à l'intérieur de **Liste2**.
- La procédure qui permet d'établir la troisième catégorie de liens consiste à chercher les liens qui existent, à une itération donnée, entre l'élément d'exclusion (définie au chapitre 3.3.2.2) et les noeuds de **Liste2**.

2.8.3.1 - Liens entre listes (entre Liste1 et Liste2)

La liaison entre les noeuds de **Liste1** et de **Liste2** se fait à partir de la procédure suivante :

```
1   Liaison_inter_liste(pointeur :Liste1, pointeur :Liste2, Région : nouveau_trapèze)
2   {
3    nombre_liens_Liste1=0;
4    Tant que(Intersection(Liste1→Région, nouveau_trapèze) == faux et Liste1!=NULL)
5      Liste1 = Liste1→suivant;
6      Intersection(Liste1→Région, nouveau_trapèze) ;
7        Si(Liste1→lien[nombres_liens_Liste1] != NULL)
8              nombres_liens_Liste1 = nombres_liens_Liste1 + 1;
9        Liste1→lien[nombres_liens_Liste1]=Liste2;
10       Liste1→lien[nombres_liens_Liste1 + 1]=NULL;
11       Liste2→lien[0]=Liste1;
12       Liste2→lien[1]=NULL;
13       Si(Liste1→suivant != NULL)
14             Liaison_inter_liste(Liste1→suivant, Liste2→suivant, nouveau_trapèze);
     }
```

Voici en détail l'explication de la procédure :

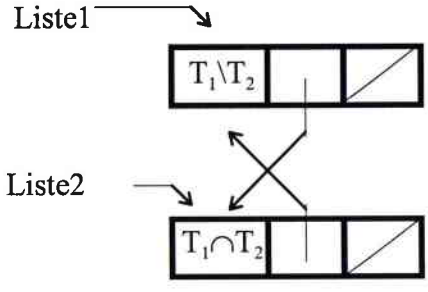
- Ligne 3 → Initialisation de paramètres représentant le nombre de liens pour chaque noeud en cours de traitement de **Liste1**.
- Lignes 4-6 → En parcourant **Liste1**, on vérifie s'il y a intersection entre la région représentée par le noeud actuel de **Liste1,** et entre un nouveau trapèze associé à un segment d'obstacle. Tant que l'intersection n'est pas vérifiée pour un noeud de **liste 1** (et que le dernier noeud de **Liste 1** n'a pas été atteint), on passe au noeud suivant de **Liste1**, jusqu'à ce qu'une intersection soit trouvée.

- Lignes 7-8 → Avant d'ajouter un nouveau lien au tableau des liens du noeud de **Liste1**, on vérifie s'il n'y a pas des liens qui ont déjà été réalisés.
- Lignes 9-12 → On établit les liens entre le noeud en cours de traitement de **Liste2** et le noeud en cours de traitement de **Liste1**
- Lignes 13-14 → Tant que le dernier noeud de **Liste1** n'a pas été atteint, on passe au noeud qui suit le noeud trait. On procède de même pour **Liste2**.

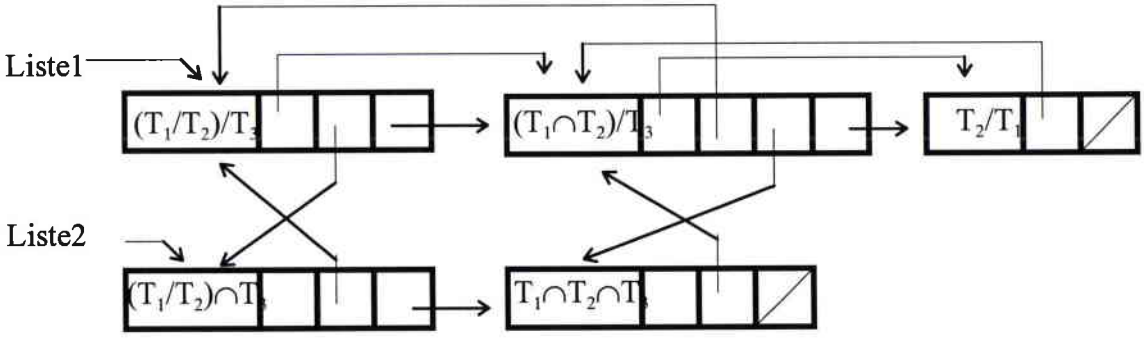
Exemple : Supposons avoir trois trapèzes T_1, T_2, T_3 :

Itération 1 : T_1 et T_2 présentent une intersection :

- on met dans **Liste1** le complément de T_1 par rapport à T_2 : $T_1 \backslash T_2$.
- on met dans **Liste2** l'intersection de T_1 avec T_2 : $T_1 \cap T_2$.



Itération 2 : T_3 présente une intersection avec T_1/T_2 et $T_1 \cap T_2$ mais pas avec T_2/T_1



Liste1 contient trois noeuds, même s'il n'y a aucune intersection.

Liste2 contient deux noeud, puisqu'on a intersection de T_3 uniquement avec T_1/T_2 et $T_1 \cap T_2$.

2.8.3.2 - Liens à l'intérieur d'une liste (Liste2)

Pour établir les liens entre les propres noeuds de **Liste2**, on se sert d'une part, des liens entre les noeuds de **Liste1** et entre ceux de **Liste2**, et d'autre part des liens entre les propres noeuds de **Liste1**.

Ces liens se font à partir de la procédure suivante :

```
1  LiensListeCopie(pointeur :Liste1, pointeur :Liste2, Région: nouveau_trapèze)
2  {
3      nombre_liens_Liste1 = 0;
4      nombre_liens_suivant = 0 ;
5      j=1 ;
6      Tant que(Intersection(Liste1→Région, nouveau_trapèze)==faux et Liste1 != NULL)
7          Liste1 = Liste1→suivant;
8      Si( Liste1 != NULL)
9          Tant que(Liste1→lien[y]!= NULL)
10             nombre_liens_Liste1 = nombre_liens_Liste1 + 1;
11     Si(nombre_liens_Liste1>1)
12         Pour(i=0;i<nombre_liens_Liste1;i=i+1)
13             nombre_liens_suivant = 0;
14             Si(Liste1→lien[i]→lien[nombre_liens_suivant] != NULL)
15                 nombre_liens_suivant = nombre_liens_suivant + 1;
16                 Si(Intersection(Liste1→lien[i]→Région, nouveau_trapèze)==vrai)
17                     Liste2→lien[j] = Liste1→lien[i]→lien[nombre_liens_suivant-1];
18                     j = j + 1;
19     Si(Liste1→suivant != NULL)
20         LiensListeCopie(Liste1→suivant, Liste2→suivant, nouveau_trapèze);
   }
```

Voici en détail l'explication de cette procédure :

- Lignes 3-5 → Initialisation de paramètres.
- Lignes 6-7 → En parcourant **Liste1**, on vérifie s'il y a intersection entre la région représentée par le noeud actuel de **Liste1,** et entre un nouveau trapèze associé à un segment d'obstacle. Tant que l'intersection n'est pas vérifiée pour un noeud de **liste1** (et que le dernier noeud de **Liste1** n'a pas été atteint) , on passe au noeud suivant de **Liste1**, jusqu'à ce qu'une intersection soit trouvée.
- Lignes 8-10 → Tant que le dernier noeud de **Liste1** n'a pas été atteint, on calcule, à chaque nouvelle itération, le nombre de liens de chaque noeud successif de **Liste 1**
- Lignes 11-18 → Dans la procédure en question, nous avons désigné par **Liste 1→ lien[i]** tout noeud de **liste1** ayant un lien avec le noeud en cours de traitement dans **Liste1** , et par **Liste2 → lien[j]** , pour $j > 0$, tout noeud de **Liste2** lié au noeud en cours de traitement dans **Liste 2 .** A condition que la région représentée par tout noeud **Liste1→ lien[i]** présente une intersection avec le nouveau trapèze introduit, tout noeud **Liste2→ lien[j]** correspond au noeud de **Liste1** ayant un lien avec le noeud **Liste1→ lien[i].**
- Lignes 19-20 → Tant que le dernier noeud de **Liste1** n'a pas été atteint, on passe au noeud suivant. On passe aussi au noeud suivant de **Liste2.**

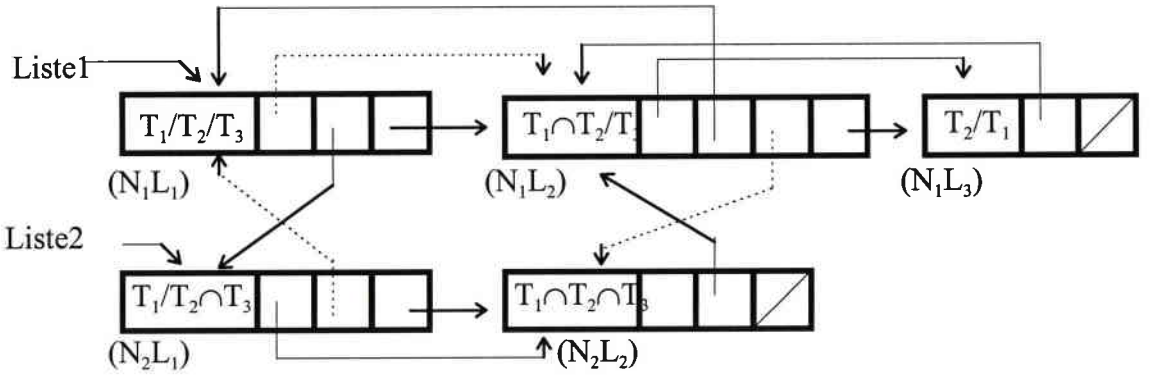
Notons que l'indice **[j]** représente une séquence régulière de nombres qui se suivent (1, 2, 3, 4 etc..) alors que **[i]** dépend de la condition sus mentionnée, et ne représente donc pas une séquence régulière de nombres.

Exemple:

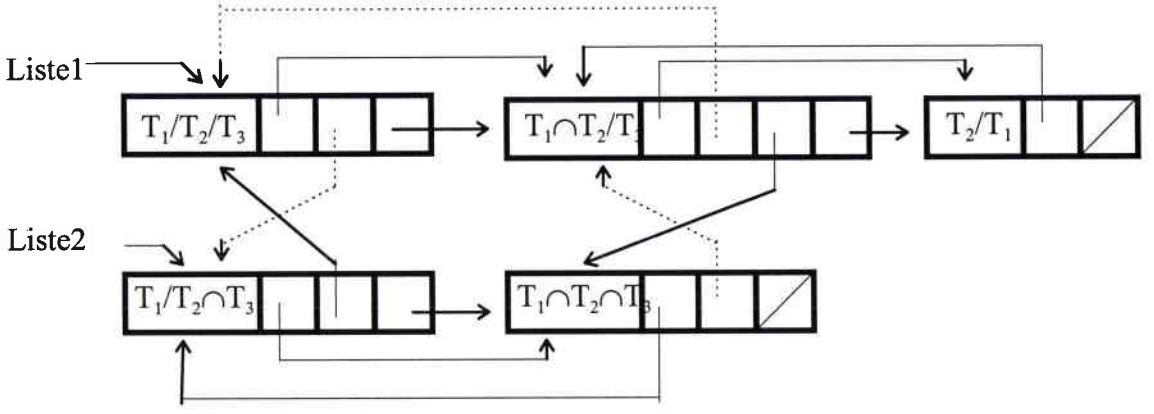
Reprenons l'exemple précèdent, et nos trois trapèzes T_1, T_2 et T_3, ainsi que le résultat de l'exemple précèdent sur le contenu de **Listes1** et de **Liste2.**

Pour trouver le lien du premier noeud (N_1L_2) de **Liste2** représentant la région $\mathbf{(T_1/T_2)\cap T_3}$, on cherche d'abord le noeud (N_1L_1) de **Liste1** représentant $\mathbf{(T_1/T_2)/T_3}$, auquel ce premier noeud est lié. On cherche ensuite le noeud lié à (N_1L_1) dans **Liste1**, c. à. d. le noeud (N_2L_1), qui représente la région $(T_1\cap T_2)/T_3$. Puis de (N_1L_1), on cherche dans **Liste2** le noeud auquel ce dernier est lié. Nous trouvons ainsi le noeud (N_2L_2) de **Liste 2**, qui est lié au noeud (N_1L_2) de

la même liste. Le pointillé indique le chemin parcouru par les pointeurs pour établir le lien du premier noeud de Liste2 au sein de la même liste.



On procède de la même façon pour établir le lien du deuxième noeud de **Liste2** à l'intérieur de la même liste.



2.8.3.3 - Liens de l'élément d'exclusion avec d'autres noeuds

Pour pouvoir établir les liens de l'élément d'exclusion, on numérote tous les noeuds de **Liste1** et de **Liste 2**. Pour **n** segments, les noeuds de **Liste1** vont être numérotés de 1 à $2^{n-1}-1$. Ceux de **Liste 2** de 2^{n-1} à 2^n-2. L'élément d'exclusion aura le dernier numéro 2^n-1.

Soit une variable : $\mathbf{m = 2^{n-1} - 1}$

On établit les liens de l'élément d'exclusion avec les noeuds contenant les numéros $m + 2^0$, $m + 2^0 + 2^1$, ..., $m + 2^0 + 2^1 + ... + 2^{n-2}$.

Un élément d'exclusion étant obtenu à chaque itération, celui obtenu à la $i^{ème}$ itération va représenter la région contenant uniquement l'information du $i^{ème}$ segment. Cette région fait partie du nouveau trapèze introduit à la $i^{ème}$ itération, sans faire partie des régions constituées jusque là à partir des intersections des trapèzes introduits au cours des itérations précédentes. Cette région correspond donc à $T_i \setminus (T_{i-1} \setminus (....\setminus(T_1)...))$.

Du point de vue informatique, les régions de tous les éléments d'exclusion sont identifiées a priori. Les codes représentant ces régions sont mis dans une liste appelée : « Liste_Régions_indépendantes ».

Pour pouvoir déterminer les régions représentées par les éléments d'exclusion:

a- On calcule l'union de tous les trapèzes introduits au cours des itérations précédant la $i^{ème}$ itération.

b- On prend le complément de cette union par rapport au trapèze introduit à la $i^{ème}$ itération.

Exemple :

Supposons avoir 4 trapèzes T_1, T_2, T_3 et T_4.

Les régions pour chaque membre de " Liste_Régions_indépendantes " vont être constituées de la manière suivante :

- élément 1 : T_1 (premier trapèze, pas d'antécédents, pas d'union)
- élément 2 : $T_2 \setminus T_1$ (un trapèze qui précède T_2, pas d'union)
- élément 3 : $T_3 \setminus (T_1 \cup T_2)$
- élément 4 : $T_4 \setminus (T_1 \cup T_2 \cup T_3)$

Voici la procédure qui permet d'établir les liens de l'élément d'exclusion :

```
1  Inserfin_élement_exclusion(pointeur : Liste1 ,Région : Région_indépendante)
2  {
3   élément_d'exclusion→Région = Copie(Région_indépendante);
4   élément_d'exclusion→numéro = 2×m+1;
5   Tant que(Liste1→numéro < m)
6         Liste1 = Liste1→suivant;
7   Si(Liste1 != NULL)
8         nombre_sauts=0;
9         nombre_liens_exclusion=0;
10        Tant que(nombre_sauts<nombre_segments-1)
11              déplacement_noeud = déplacement_noeud + 2^nombre_sauts.
12              Tant que((Liste1→numéro < (déplacement_noeud + m)) &&
                Liste1→suivant !=NULL)
13                    Liste1 = Liste1→suivant;
14              Si(déplacement_noeud ≤ m)
15                Alors Si(Liste1→numéro == déplacement_noeud + m)
16                      élément_exclusion→lien[nombre_liens]=Liste1;
17                      compteur =0;
18                      Tant que(Liste1→lien[compteur]!=NULL){compteur = compteur+1}
19                            Liste1→lien[compteur] = élément_exclusion;
20                            Liste1→lien[compteur+1] = NULL;
21                      nombre_liens = nombre_liens + 1;
22        nombre_sauts = nombre_sauts + 1;
23   élément_d'exclusion→lien[nombre_liens] = NULL;
24   élément_d'exclusion→suivant = NULL;
25   dernier(Liste1)→suivant = élément_exclusion;
     }
```

La définition de cette procédure est réalisée comme suit :

- Lignes 3-4 → On attribue à l'élément d'exclusion, comme numéro: la valeur 2m+1, et comme adresse de région : un membre correspondant à une itération donnée de la "Liste_Régions_indépendantes".
- Lignes 5-6 → On cherche le noeud de **Liste1** qui a le numéro **m**. Tant que le numéro d'un noeud de **Liste1** est plus petit que **m**, on passe au noeud suivant. Toute la partie de **Liste1** jusqu'au numéro **m,** correspond à **Liste1** avant d'y insérer **liste2**. (**Liste1-**).
- Lignes 7-13 → Soit **nombre_sauts** une variable variant de 0 à nombre de segments-2, et soit **déplacement_noeud** une suite géométrique de coefficient 2 et de dimension **nombre_sauts**. Pour pouvoir trouver les noeuds de **Liste1** qui ont des liens avec l'élément d'exclusion, on cherche les noeuds qui ont des numéros identiques aux différentes valeurs de **déplacement_noeud.**
- Lignes 14-24 → Etablissement des liens réciproques entre l'élément d'exclusion et le noeud actuel de **Liste1** qui a un numéro identique à **déplacement_noeud**.
- Ligne 25 → Insertion de l'élément d'exclusion à la fin de **Liste1**.

2.8.4 - Insertion d'une liste

Pour insérer **Liste2** à la fin de **Liste1**, dans le but d'avoir une seule liste, on utilise la procédure suivante :

```
Inser_fin_Copie(pointeur : Liste1, pointeur : Liste2)
{
    Dernier(Liste1)→ suivant = Liste2;
}
```

Dernier étant une fonction qui permet d'accéder au dernier noeud d'une liste (**Liste1** dans notre cas).

La fonction **Dernier** est défini par :

```
Dernier(pointeur : liste)
{
    Tant que(liste != NULL)
        liste = liste→suivant;
    retourne(liste);
}
```

A chaque itération, **Liste2** est insérée à **Liste1**, ensuite l'élément d'exclusion est inséré à l'ensemble (voir figure 2.28), on obtient ainsi de manière récursive une nouvelle liste **Liste1** dans l'itération suivante.

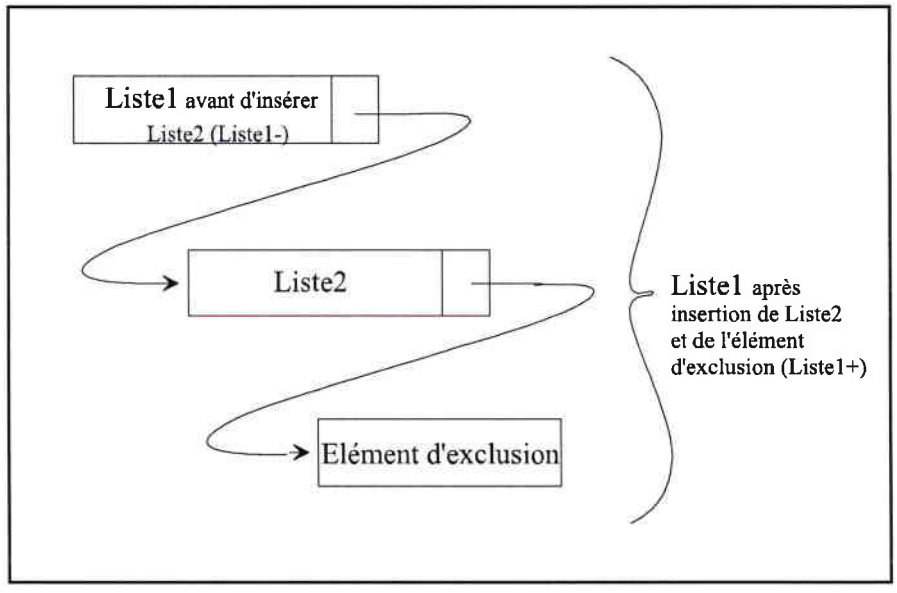


Figure 2.28

Utilisation de Liste1 de manière récursive

2.8.5 - Suppression d'un noeud du graphe

On peut être amener à enlever un noeud du graphe qui représente une région suffisamment petite telle que le robot ne puisse pas la traverser.

La procédure qui permet d'effacer un tel noeud est la suivante :

(1) D'abord on cherche dans la liste des noeuds, le noeud précédent celui qu'on veut enlever **NoeudPrécédent**.

(2) Ensuite, on fait pointer **NoeudPrécédent** vers le noeud suivant le noeud à enlever. Enfin on fait pointer le noeud à enlever vers NULL (voir figure 2.29). De cette façon, le noeud en question ne fait plus partie du graphe.

(3) A la fin, on efface de la mémoire.

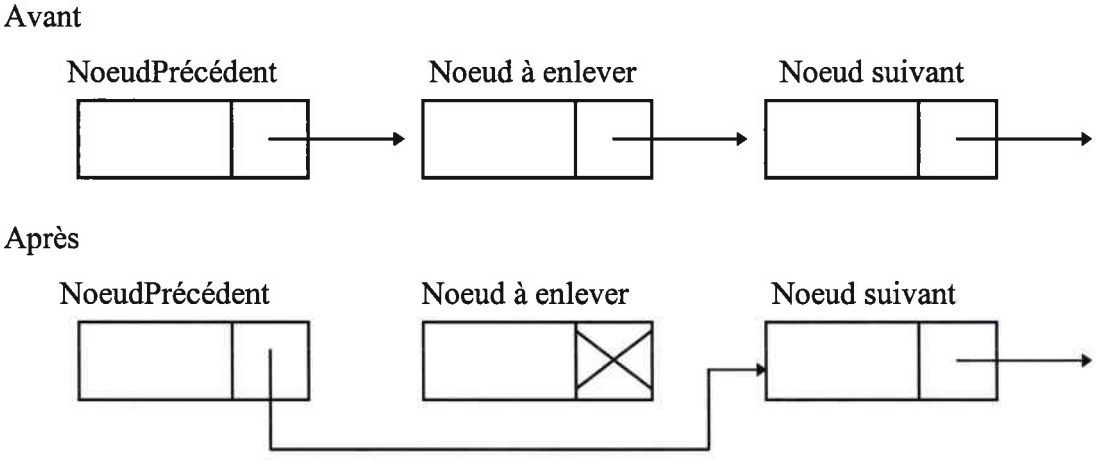


Figure 2.29 Procédure pour enlever un noeud d'une liste

2.9- Acquisition des informations sur les segments d'obstacles

Chaque noeud, représentant une région issue des intersections entre trapèzes, doit contenir les informations (orientations $\theta°_1$, $\theta°_2$, ..., $\theta°_n$ définies précédemment) sur les segments d'obstacles détectés par le robot dans cette région. Ces informations sont stockées dans un des membres du noeud, qu'on appellera " tableau d'informations ".

Appelons "**Liste1-**": **Liste1**, avant d'y insérer **Liste2** et appelons "**Liste1+**" : **Liste1**, après l'insertion de **Liste2**.

La répartition des informations à chaque itération i est la suivante :

- Pour la première itération (initialisation de Liste1), **Liste1** est constituée d'un noeud qui contient l'information $\theta°_1$ sur le segment $\mathbf{s_1}$.
- Les noeuds de **Liste1-** contiennent les informations de **Liste1+** de l'itération précédente. (de $\theta°_1$ jusqu'à $\theta°_{i-1}$)
- Les noeuds de **Liste2** contiennent les informations des noeuds de **Liste1-** avec lesquelles elles ont un lien. De plus, tous les noeuds de **Liste2** contiennent l'information $\theta°_i$.
- L'élément d'exclusion contient uniquement l'information $\theta°_i$.

Exemple :

Première itération : Liste1+ contient trois noeuds représentant les régions $\mathbf{T_1 \backslash T_2}$, $\mathbf{T_1 \cap T_2}$ et $\mathbf{T_2 \backslash T_1}$ avec respectivement les informations $(\theta°_1)$, $(\theta°_1$ **et** $\theta°_2)$ et $\theta°_2$.

Deuxième itération : Liste1- contient trois noeuds représentant les régions $\mathbf{(T_1 \backslash T_2) \backslash T_3}$, $\mathbf{(T_1 \cap T_2) \backslash T_3}$ et $\mathbf{(T_2 \backslash T_1) \backslash T_3}$, avec respectivement les mêmes informations contenues dans **Liste1+** de la première itération. Comme $\mathbf{T_3}$ ne présente une intersection qu'avec la région $\mathbf{T_1 \backslash T_2}$, **Liste2** ne contient qu'un seul noeud, représentant la région $\mathbf{(T_1 \backslash T_2) \cap T_3}$). Ce noeud possède 2 informations : l'information $(\theta°_1)$ du noeud, représentant la région $\mathbf{(T_1 \backslash T_2) \backslash T_3}$), auquel il est lié dans **Liste1+**, et l'information $(\theta°_3)$ obtenue dans $\mathbf{T_3}$. L'élément d'exclusion contient uniquement l'information $(\theta°_3)$. Après avoir construit le graphe de noeuds, plusieurs chemins entre les régions sont possibles. Il reste à choisir le chemin optimal.

2.10 - Application à la planification de chemins

2.10.1 - Utilisation de l'algorithme de Dijkstra

Il serait intéressant de connaître, parmi tous les chemins qui relient un noeud source à un noeud but à travers les liens existant entre les noeuds du graphe, celui ou ceux qui permettrons au robot de se diriger vers son but de la meilleure façon possible, du point de vue **robustesse**. On considère que moins il y aura de changement d'informations capteurs sur un chemin, plus ce chemin sera robuste. Or à chaque fois que le robot passe d'une région à une autre, il y a changement d'information. Le chemin optimal va correspondre à celui où le robot passe par le minimum de régions avant d'atteindre son but.

On utilise l'algorithme exhaustif de Dijkstra [Dij 59] pour trouver le chemin optimal. L'algorithme qu'on a choisit ne nécessite pas l'utilisation de fonctions heuristiques comme c'est le cas de l'algorithme A*.

Rappelons que le graphe de noeuds qu'on a construit est un ensemble de sommets S et ensemble d'arcs A. Les arcs $\mathbf{a} \in \mathbf{A}$ sont munis de coûts $\mathbf{d(a)} \geq \mathbf{0}$. d(a) = 1 entre deux sommets adjacents. On cherche les coûts minimums (ici le minimum de régions) entre un sommet $\mathbf{x^\circ}$ et les autres sommets du graphe.

Le principe de l'algorithme consiste à ajouter, étape par étape, à l'ensemble $\mathbf{X} = \{\mathbf{x^0}\}$, un sommet **y** pour lequel le coût entre celui-ci et $\mathbf{x^\circ}$ soit le plus petit.

Pour chaque sommet **x**, on désignera :

- par **l(x)** le coût provisoire le plus petit entre $\mathbf{x^\circ}$ et **x**.
- et par **précédent(x)** le sommet précédant **x.** Le coût entre précédent(x) et $\mathbf{x^\circ}$ est minimal

A chaque étape, on notera X = {x} dont le coût le plus faible **l(x)** a définitivement été calculé, et par Y, son complémentaire par rapport à l'ensemble des sommets S. Soit P(x) l'ensemble des sommets voisins de x et $d(x_1, x_2)$ le coût entre deux sommets x_1 et x_2.

Initialement :

- On pose pour tout $x \neq x^\circ$: $l(x) = \infty$, précédent(x) = -1, et $l(x^\circ) = 0$.
- On prend $X = \{x^\circ\}$, $Y = S \setminus \{x^\circ\}$. Pour tout $y \in Y \cap P(x^\circ)$, on pose l(y) = d(x, y), précédent(y) = x°.

Puis on réalise l'itération suivante :

```
Procédure Dijkstra()
{
   Tant que Y≠Nil,
              et qu'il existe dans Y au moins un y tel que l(y) ≤ ∞,
                prendre un des y de Y tel que l(y) soit minimum
                pour chaque z de Y ∩ P(y),
                  Si l(z) > l(y) + d(y, z) :
                     Alors l(z) = l(y) + d(y, z)
                     précédent(z) = y .
   X = X ∪{y} ; Y = Y\{y} ;
}
```

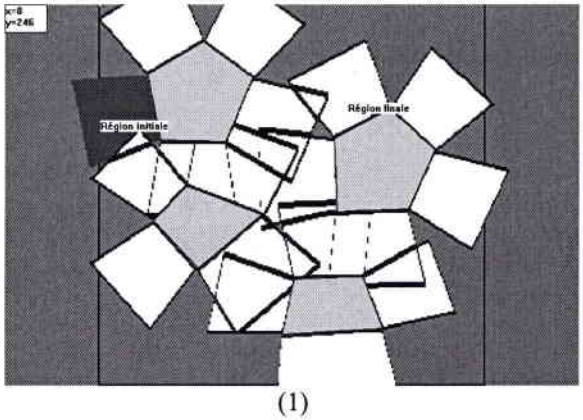
Lorsque l'algorithme s'arrête, l'ensemble **X** contient tous les sommets **x** tels que **l(x) < ∞**, c. à. d. ceux dont la plus courte distance à x° peut être calculée, et **Y** contient l'ensemble des autres sommets. **X** est donc l'ensemble des sommets extrémités des chemins issus de **x°**, et **Y** est constitué des sommets qu'on ne peut pas atteindre par un chemin issu de **x°.**

L'algorithme donne donc non seulement les sommets accessibles depuis **x°**, mais aussi les coûts des chemins optimaux.

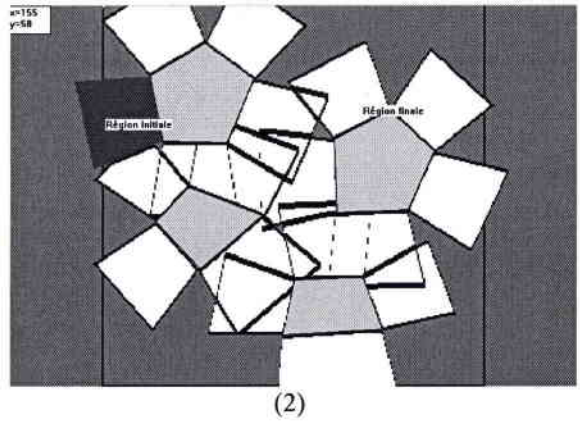
L'algorithme de Dijkstra a une complexité en temps en $O(n^2)$ où n est le nombre de sommets (de noeuds) d'un graphe. Il est intéressant puisqu'il ne nécessite pas le développement de l'ensemble des chemins. Par contre l'ensemble des noeuds doit être atteint avant d'affirmer que la solution optimale a été trouvée. On ne peut pas utiliser l'algorithme A* car on n'a pas à notre disposition une fonction heuristique qui permet d'estimer le coût entre un noeud N et le noeud but.

2.10.2 - Résultats

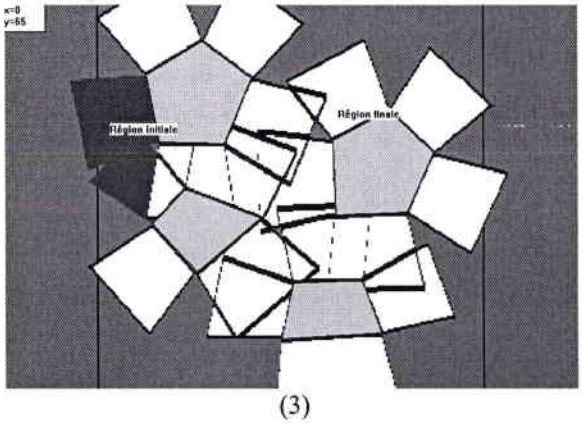
On présente ci-dessous l'évolution du chemin à travers les sous-régions créées à partir des informations capteurs, dans le cas d'obstacles polygonaux et d'obstacles rectangulaires. La progression du chemin d'une région initiale jusqu'à la région finale est indiquée par chaque sous-région empruntée par le chemin, qui est coloriée en gris sombre. La largeur d_m d'un trapèze associé à un segment d'obstacle est de 50 pixels et l'angle d'ouverture du trapèze θ_s = 18 degrés.



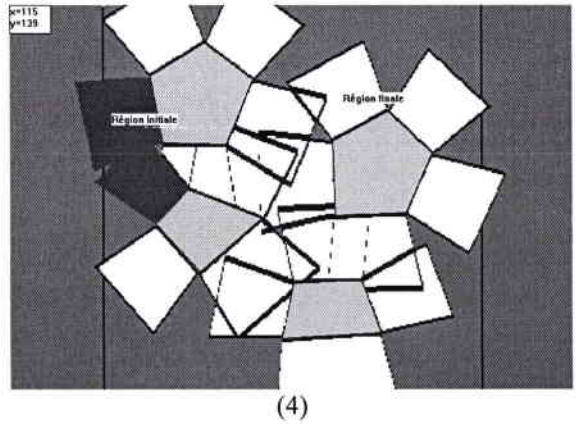
(1)



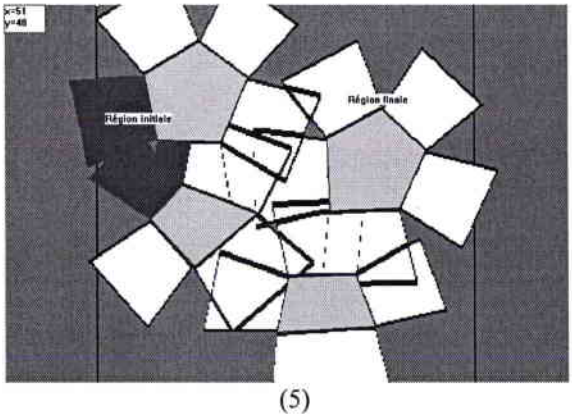
(2)



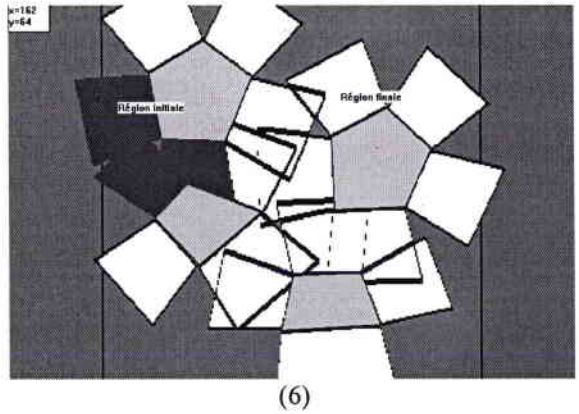
(3)



(4)



(5)



(6)

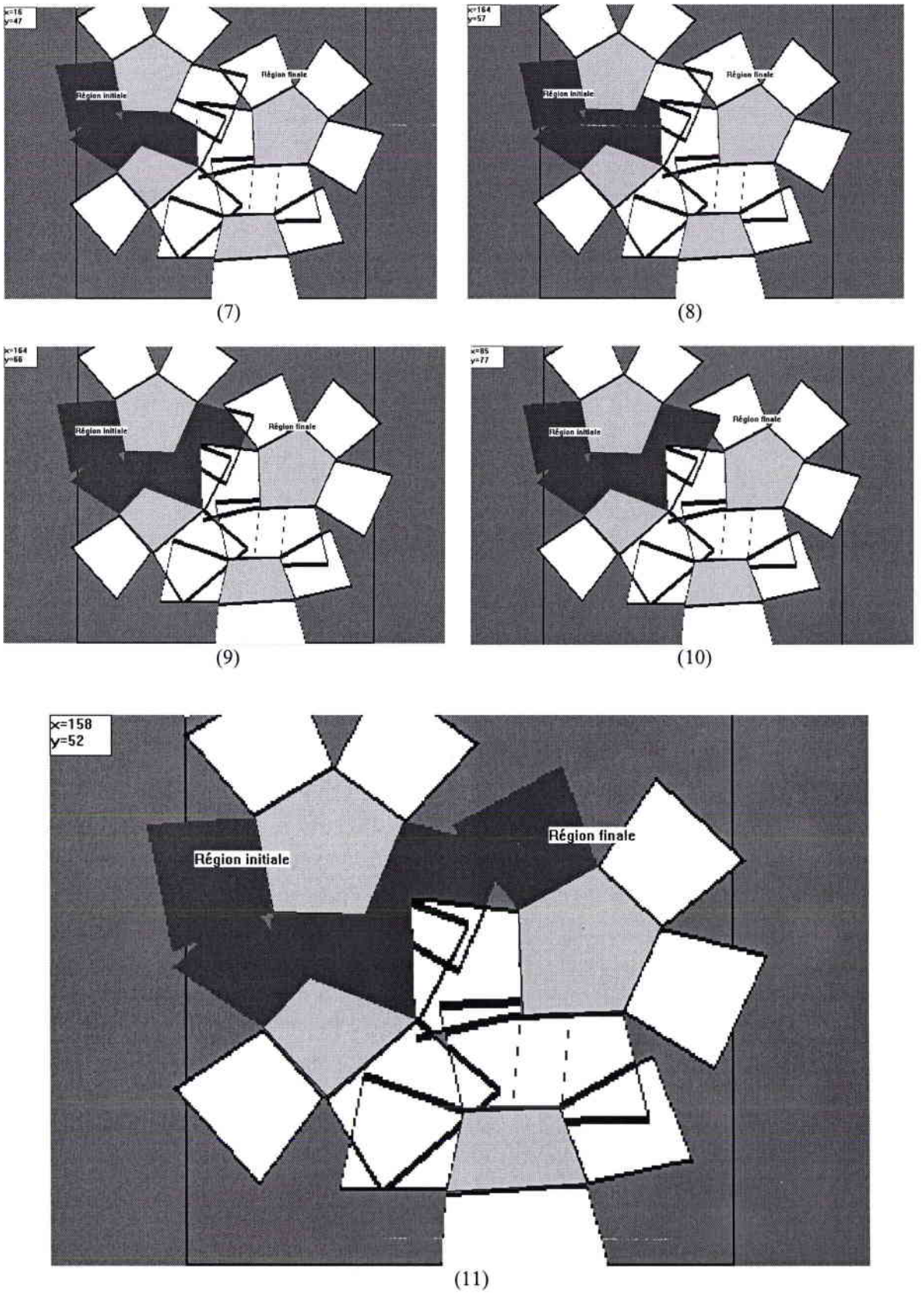
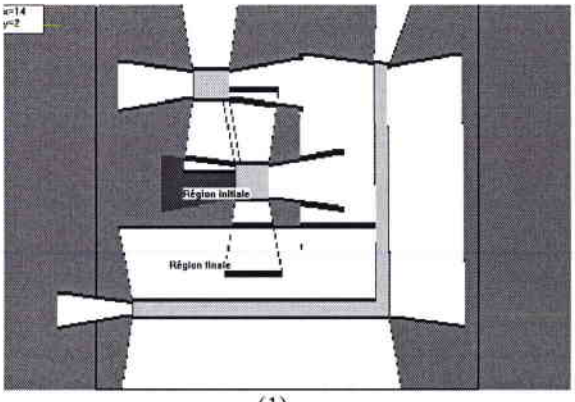
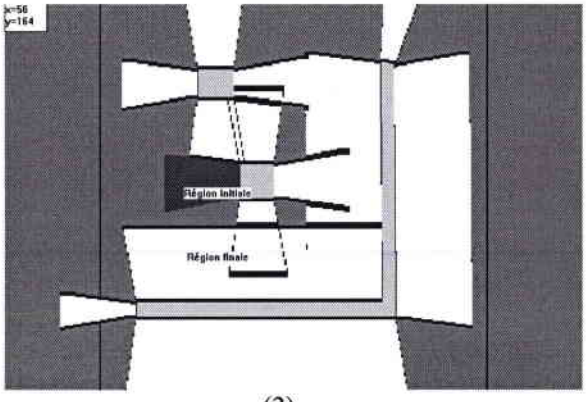


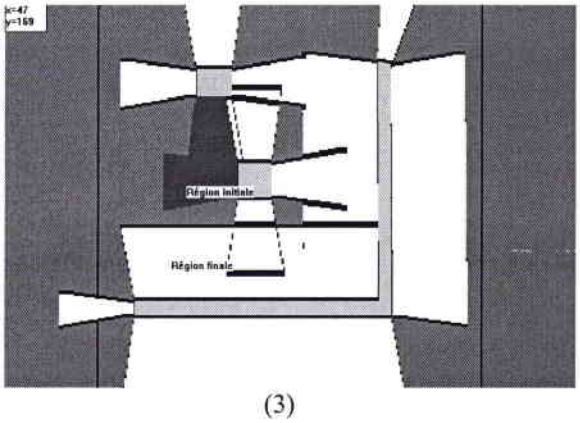
Figure 2.30
Résultats obtenus sur l'évolution du chemin d'une région initiale
jusqu'à une région finale en 11 étapes, dans le cas d'obstacles polygonaux.


Région initiale
Région finale

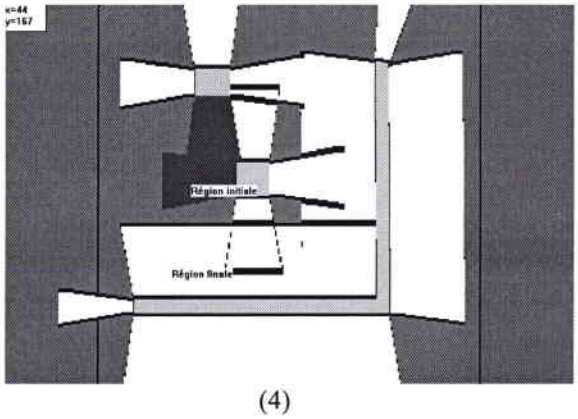
(1)


Région initiale
Région finale

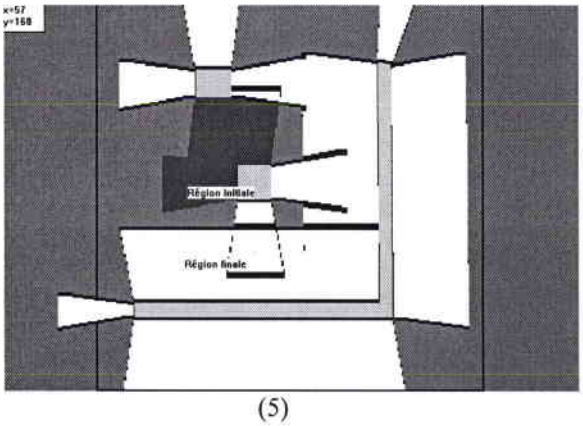
(2)


Région initiale
Région finale

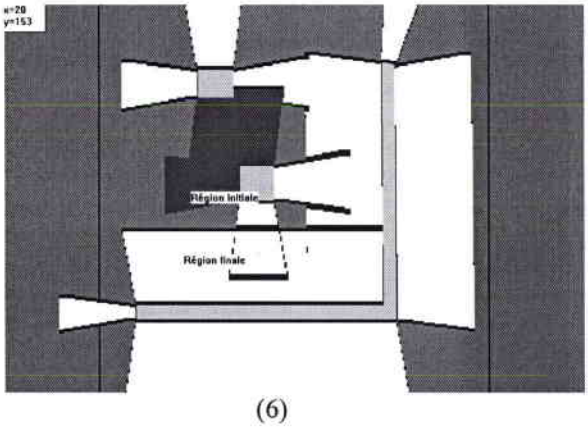
(3)


Région initiale
Région finale

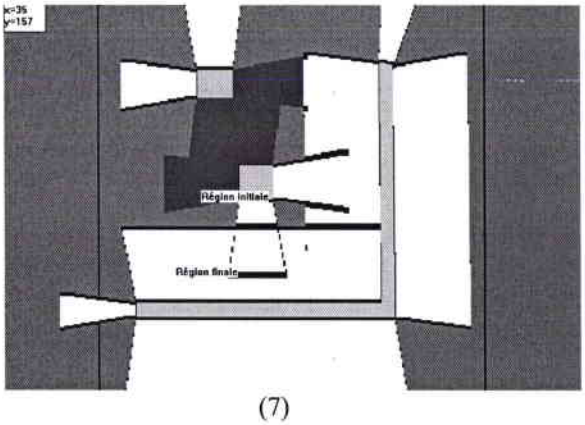
(4)


Région initiale
Région finale

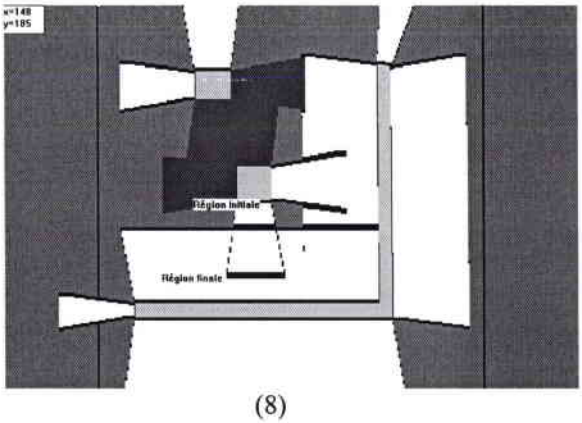
(5)


Région initiale
Région finale

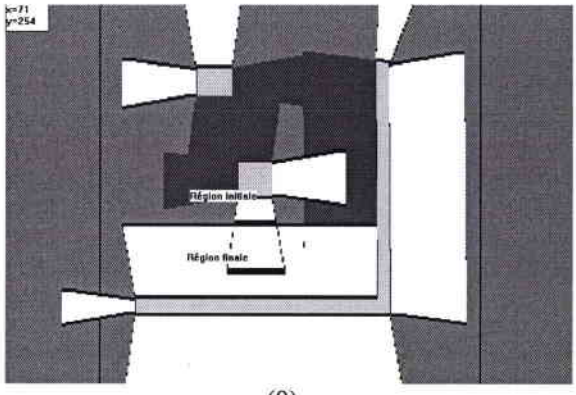
(6)


Région initiale
Région finale

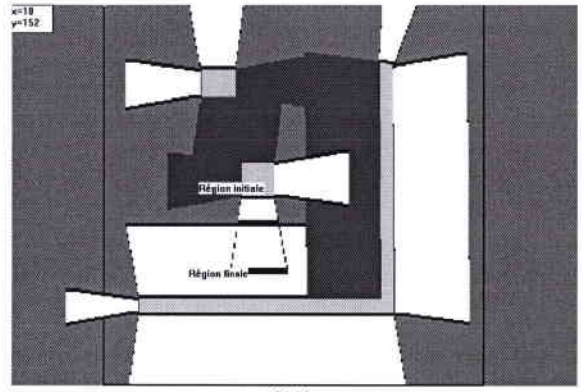
(7)


Région initiale
Région finale

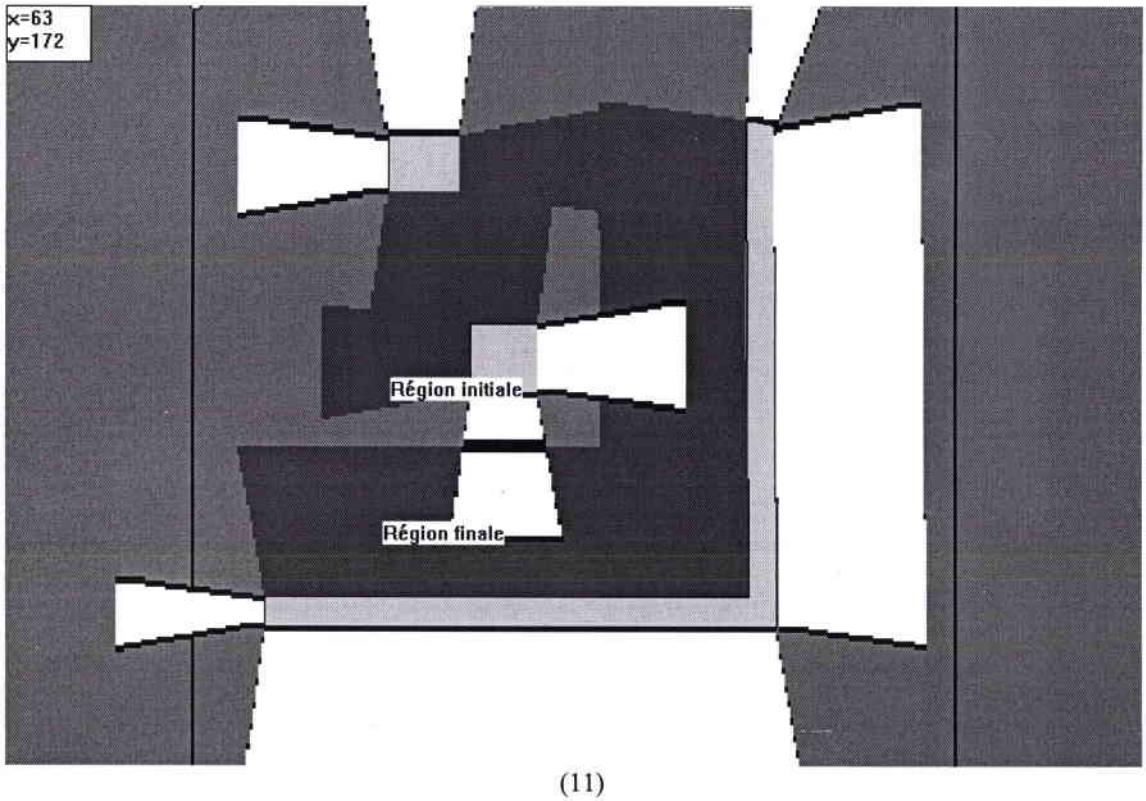
(8)



(9)



(10)



(11)

Figure 2.31
Résultats obtenus sur l'évolution du chemin d'une région initiale
jusqu'à une région finale en 11 étapes, dans le cas d'obstacles rectangulaires.

2.11 - Conclusion

Dans ce chapitre, nous avons montré comment l'environnement a été modélisé. Le modèle créé ne repose pas sur des contraintes géométriques comme c'est le cas des méthodes géométriques de modélisation, mais plutôt sur l'observation que le robot a de l'environnement. Cette observation est effectuée lorsque le robot acquière des informations propres et spécifiques à chaque segment d'obstacle de l'environnement. Les régions qui sont issues de la modélisation constitue chacune une zone où le robot peut se repérer par rapport aux orientations dans lesquelles sont effectuées les mesures du capteur.

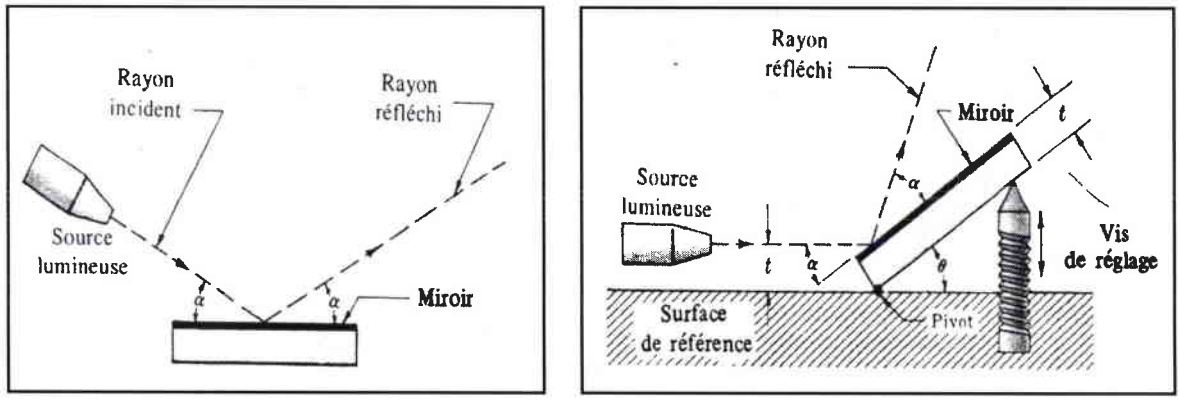
Le temps de modélisation sur un PC *pentium* 100Mhz, avec la méthode utilisée, est de 0.55s pour l'environnement de la figure 2.30 et de 0.11s pour celui de la figure 2.31. Le temps de planification pour les deux cas de figures est très faible et inférieur à 0.01s.

CHAPITRE III

Systèmes de commande du robot

Systèmes de commande de la trajectoire du robot

Dans le langage moderne, le sens du mot **système** est devenu ambigu. Du point de vue abstrait, un **système** est un assemblage, un ensemble ou une collection d'objets reliés ou branchés les uns aux autres de façon à former une entité ou un tout. Du point de vue plus spécifique par rapport à la littérature scientifique, un **système** est un assemblage de constituants branchés ou reliés de telle façon qu'ils forment une entité individualisée et/ou agissant comme telle. Dans le domaine de la technologie et de la science, un **système de commande** est un système qui permet de se commander, se diriger ou se régler lui-même, ou bien commander, diriger ou régler un autre système de manière active et dynamique. Le miroir représenté à la figure 3.1a peut être considéré comme un système de commande élémentaire réglant le rayon lumineux selon l'équation simple : « l'angle de réflexion est égal à l'angle d'incidence α ». Le système représenté à la figure 3.1b constitué par un miroir pouvant pivoter sur l'une de ses extrémités et pouvant être relevé ou abaissé au moyen d'une vis à l'autre extrémité, est à proprement parler un système de commande. On règle l'angle de réflexion de la lumière à l'aide de la vis.



(a) (b)

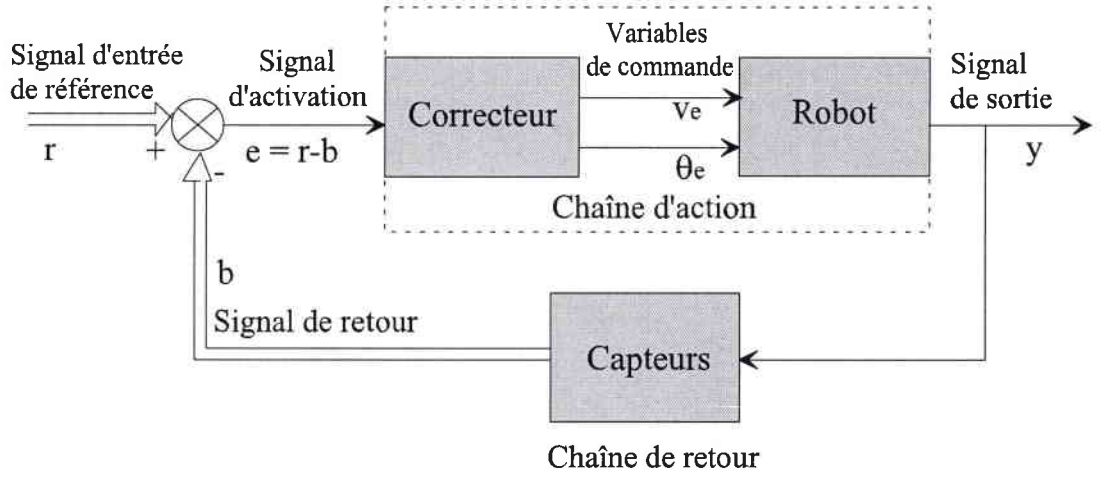
Figure 3.1

Exemple de système de commande appliqué à un miroir

3.1 - Commande du robot

3.1.1 - Principe

Le système que nous désirons commander peut se représenter sous la forme d'un schéma fonctionnel de la manière suivante :



Le signal d'entrée de référence r est constitué par l'ensemble des informations capteurs à atteindre. Le signal de retour b est constitué par les informations des capteurs mesurées en temps réel. Le signal d'activation e représente l'écart entre le signal d'entrée et celui de retour. Le correcteur permet de mettre en relation les composants de la commande u en fonction de l'écart e à minimiser. La commande du robot s'effectue à partir de la vitesse linéaire v_e et de la vitesse angulaire θ_e appliquées à son entrée (voir schéma fonctionnel).

L'équation d'état en continue du robot est de la forme :

$$\dot{x} = f(x,u) \quad \text{où } u = \begin{bmatrix} v_e \\ \theta_e \end{bmatrix}$$

Les variables de commandes sont fonctions de l'erreur e à l'entrée du système :

$$v_e = f_1(e)$$
$$\theta_e = f_2(e)$$

v_e et θ_e sont choisies de manière à faire tendre e vers 0 en fonction du temps, de façon à ce que les mesures des capteurs correspondent avec celle que l'on souhaite obtenir dans le modèle de référence du robot. A ce moment, le robot aura atteint son objectif.

3.1.2 - Calcul de l'erreur e (écart) sur les mesures

La commande du robot consiste à minimiser l'écart entre les mesures simulées dans le modèle du robot, avec les mesures obtenues dans le cas réel. Comme on a vu dans le chapitre précédent, les mesures obtenues dans le modèle du robot sont réalisées pour des orientations référencées par rapport à l'axe des abscisses du référentiel global. Alors que les mesures obtenues en temps réel sont réalisées pour des orientations référencées par rapport à l'axe du robot.

On propose de minimiser l'erreur suivante :

$$e = \int_0^{2\pi} (d(\theta) - d'(\theta - \theta_d))\delta\theta$$

où :

- θ est l'orientation à laquelle est effectuée la mesure du capteur par rapport au référentiel global (axe horizontal).
- $d(\theta)$ constitue la mesure (information sur un segment) dans le modèle du robot pour une orientation θ par rapport au référentiel global,
- $d'(\theta - \theta_d)$ la mesure du capteur dans le cas réel pour une orientation $\theta - \theta_d$ par rapport au référentiel du robot (axe du robot) (figure 3.2).
- θ_d représente l'orientation du robot par rapport au référentiel global.

e constitue l'écart entre les mesures à atteindre et celles obtenues en temps réel pour un balayage complet de l'environnement, c'est à dire de 2π.

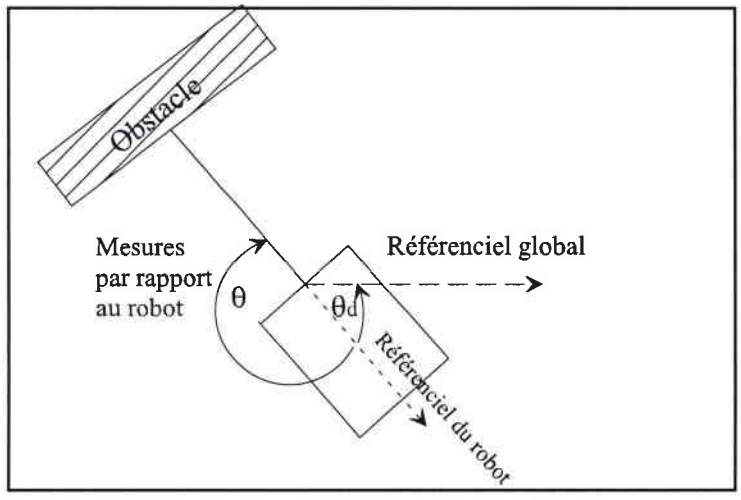


Figure 3.2 : Représentation des orientations des mesures par rapport au robot et de celui-ci par rapport au référentiel global

Pour trouver θ_d, on utilise la fonction d'autocorrélation entre le signal du modèle et celui obtenu dans le cas réel.

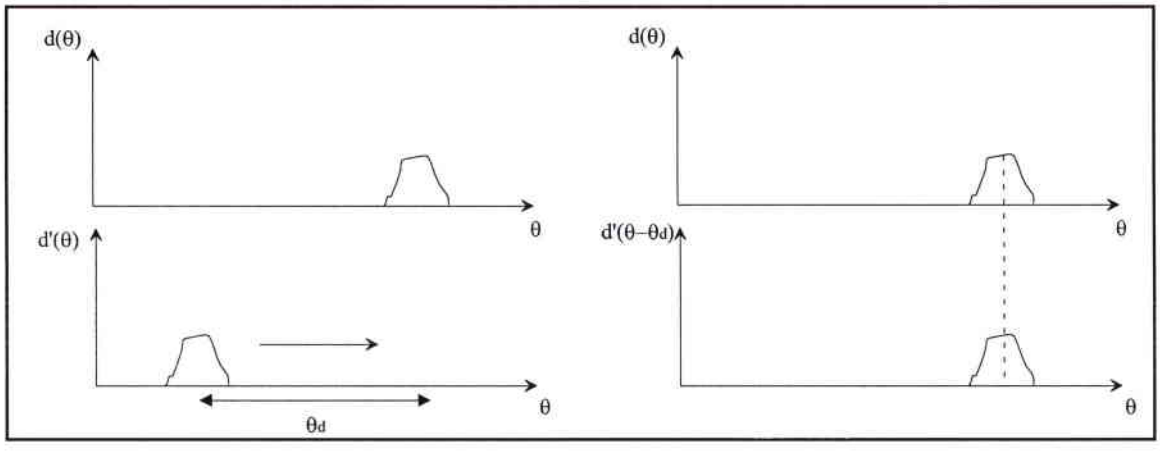


Figure 3.3. Le signal mesuré est déplacé jusqu'à ce qu'il se superpose avec le signal du modèle

La fonction d'autocorrélation est définie par l'équation :

$$P_c = \mathrm{Max}\left(\int_0^{2\pi} (d(\theta) \times d'(\theta - \theta_d) \times \delta\theta) \right)$$

Le maximum de la fonction est obtenu, en déplaçant le signal constitué par les mesures du robot dans le cas réel d'(θ - θ_d) dans un sens (on a choisi le sens trigonométrique) jusqu'à ce qu'il se superpose au signal du modèle.

3.2 - Déplacement du robot

3.2.1 - Principe

Initialement :

On utilise la fonction d'autocorrélation définie précédemment, pour obtenir l'orientation du robot par rapport au référentiel global. La variation de l'orientation du signal dans le cas réel est comprise entre 0 et 2π. On aura donc obtenu la première orientation calculée du robot θ_d par rapport à l'axe des x.

Calcul des orientations pour les configurations suivantes :

Le déplacement du robot entre deux configurations P_i et P_{i+1} est représenté sur la figure 3.4

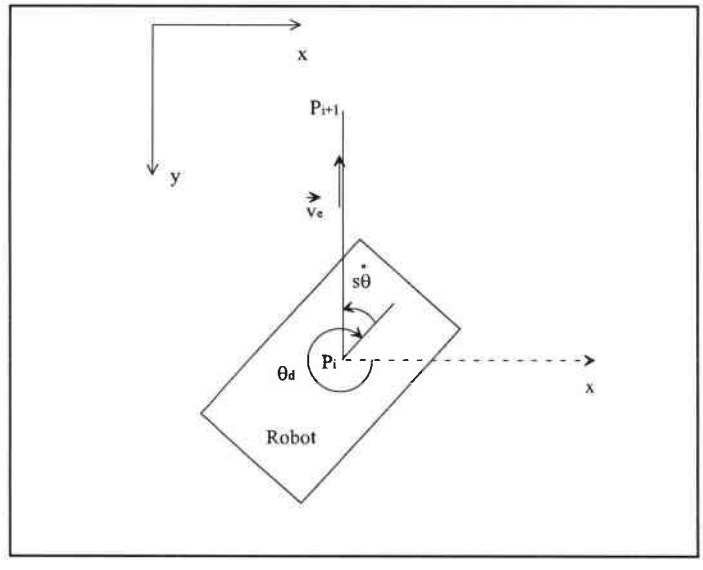


Figure 3.4

Déplacement angulaire du robot

Soit $\dot{\theta}$ la vitesse angulaire du robot.

θ_d est calculée par l'intermédiaire de la fonction d'autocorrélation dans une marge comprise entre $\theta_d - \dot{\theta}_{max}.dt$ et $\theta_d + \dot{\theta}_{max}.dt$. $\dot{\theta}_{max}$ correspond à la vitesse angulaire maximale du robot.

Soit $\dot{e}$ la dérivée première de l'écart e entre mesures. Si $\dot{e}>0$, le robot s'éloigne de la configuration but à atteindre, il se dirige dans la mauvaise direction. A ce moment, la vitesse angulaire doit être maximale (figure 3.5a), pour que le robot puisse retrouver une meilleure direction vers le point but. La commande qui a été définie précédemment consiste à minimiser l'écart entre mesures e en fonction du temps. Par contre si $\dot{e}<0$, le robot s'oriente dans la bonne direction, la vitesse angulaire du robot $\dot{\theta}$ dépend dans ce cas de la vitesse $\dot{e}$.

Soit $\ddot{e}$ la dérivée seconde à laquelle e augmente. Si $\ddot{e}>0$, cela signifie que le robot tend lentement vers le point but, on a intérêt à augmenter le vitesse angulaire $\dot{\theta}$ et à changer le signe de la vitesse pour amener le robot dans une direction d'où il peut tendre plus rapidement vers le but (figure 3.5b). Par contre si $\ddot{e}<0$, le robot tend rapidement vers la configuration but, il est orienté vers une direction idéale, et de ce fait on diminue la vitesse angulaire $\dot{\theta}$ (figure 3.5c).

Si $\dot{e}$ atteint un seuil maximal $\dot{e}_{max}$, la vitesse angulaire est à ce moment nulle. La vitesse linéaire v_e varie en fonction de la valeur de $\dot{e}$. Si $\dot{e}>0$, le robot s'éloigne du but à atteindre, v_e est égal à la vitesse linéaire minimale v_{emin} pour que le robot puisse tourner rapidement et puisse s'orienter dans la bonne direction. Par contre si $\dot{e}<0$, alors le robot se rapproche du but, et par conséquence si $v_e < v_{emax}$, où v_{emax} est la vitesse linéaire maximale autorisée, v_e augmente progressivement jusqu'à atteindre cette valeur.

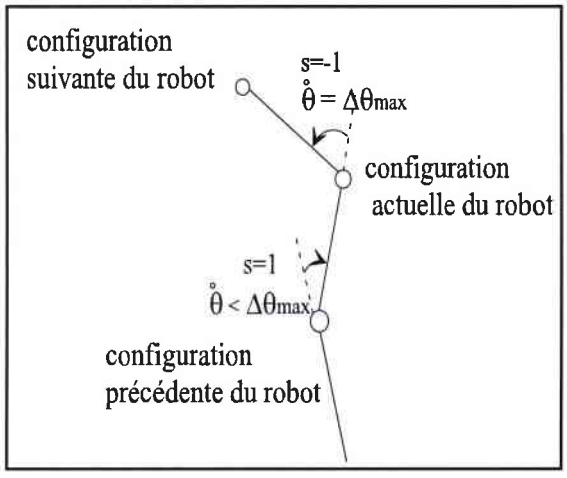


Figure 3.5 a

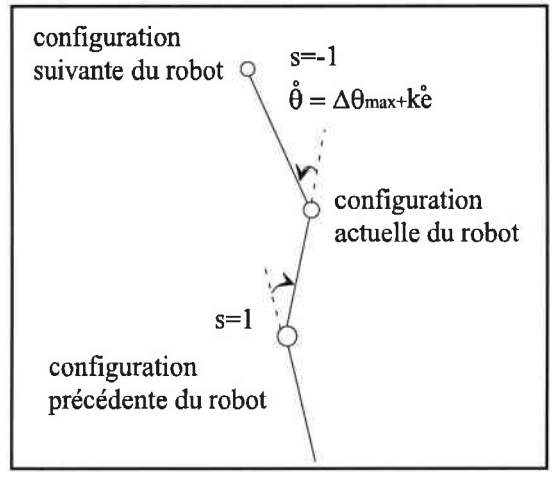


Figure 3.5b

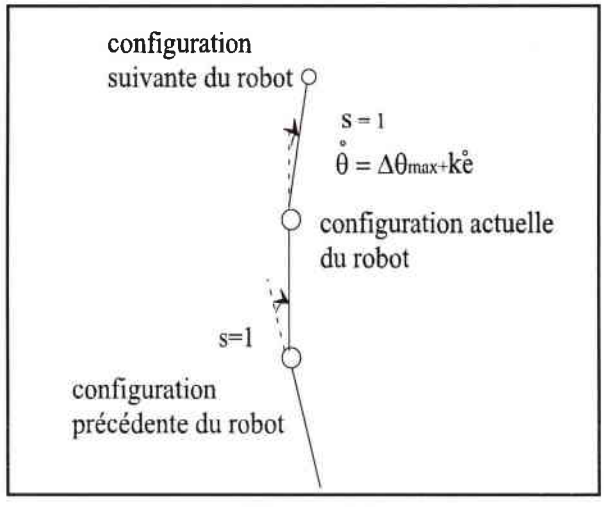


Figure 3.5c

Représentation de la variation angulaire du robot lorsque le robot s'éloigne (a), se rapproche lentement (b), et se rapproche rapidement (c) du point à atteindre.

3.2.2 - Algorithme

L'algorithme qui permet de calculer la vitesse angulaire et linéaire du robot est décrit ci-dessous, avec e_{min} la valeur minimum de e au bout de laquelle le robot est considéré comme ayant atteint le point but et k un coefficient donné.

Initialisation i=1

Tant que $e > e_{min}$

Début

Si i≠1 Passage à la configuration suivante du robot

si ($\dot{e} > 0$)

alors $\dot{\theta} = \dot{\theta}_{max}$

$v_e = v_{emin}$

sinon si ($\ddot{e} > 0$)

alors

- le robot change de sens de direction. Changement du signe de s.
- $\dot{\theta} = \dot{\theta}_{max}$
- si ($v_e < v_{emax}$) v_e augmente

sinon ($\dot{e} < 0$ et $\ddot{e} < 0$)

si ($v_e < v_{emax}$) v_e augmente

si ($\dot{e} = \dot{e}_{max} = \frac{\dot{\theta}_{max}}{k}$)

alors $\dot{\theta} = 0$

sinon

$\dot{\theta} = \dot{\theta}_{max} + k.\dot{e}$

Incrémentation i=i+1.

Fin

3.2.3 - Calcul de la trajectoire du robot en simulation

Lorsqu'on veut simuler la trajectoire du robot sur ordinateur, le déplacement est discrétisé. Le déplacement linéaire est réalisée par un pas d'une valeur donnée entre deux configurations successives.

Le déplacement du robot en coordonnées cartésiennes est le suivant :

$$x_{i+1} = x_i + v_e.\cos(\theta_d + s.\Delta\theta_i)$$
$$y_{i+1} = y_i + v_e.\sin(\theta_d + s.\Delta\theta_i)$$

où :

- x_{i+1} et y_{i+1} représentent les coordonnées du robot pour la configuration à atteindre P_{i+1},
- x_i et y_i ceux des coordonnées pour la configuration actuelle,
- v_e est une valeur qui représente le déplacement linéaire entre deux configurations successives du robot (vitesse linéaire).
- θ_d est l'orientation calculée du robot à l'étape i.
- $\Delta\theta_i$ représente le déplacement angulaire (variation angulaire, ou encore vitesse angulaire) que doit prendre le robot pour passer à la configuration suivante (étape i+1).
- s est une valeur qui représente le sens du déplacement angulaire du robot.

θ_d est calculée par l'intermédiaire de la fonction d'autocorrélation P_i discrétisée dans une marge comprise entre $\theta_d - \Delta\theta_{max}$ et $\theta_d + \Delta\theta_{max}$. $\Delta\theta_{max}$ correspond à la variation angulaire maximale du robot entre deux configurations successives (un pas).

$$P_c = \mathrm{Max}\left(\sum_{\theta_k=0}^{\theta_k=2\pi}(d(\theta_k).d'(\theta_k - \theta_d))\right)$$

L'erreur e_i entre les mesures à la configuration P_i et la configuration à atteindre est définie par :

$$e_i = \sum_{\theta_k=0}^{\theta_k=2\pi}(d(\theta_k) - d'(\theta_k - \theta_d))$$

3.2.4 - Déplacement à l'intérieur d'une région

3.2.4.1 - Cas d'un segment infini

On constate que les mesures des capteurs pour une direction donnée sont les mêmes sur toute configuration capteur située sur la même droite parallèle au segment d'obstacle. En utilisant l'algorithme sur le déplacement angulaire défini précédemment, le robot va tendre vers une droite Δ parallèle au segment d'obstacle et qui passe par la configuration à atteindre. Car la configuration capteur P_Δ pour laquelle la valeur e est minimale ($e \leq e_{min}$) se situe sur cette droite. Le robot n'est pas sûr d'atteindre le point géométrique final défini (voir figure 3.6)

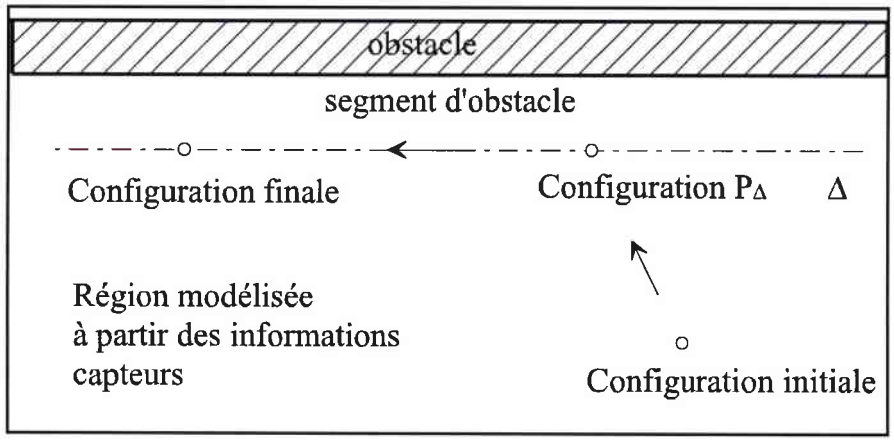


Figure 3.6 : Représentation du chemin suivi par le robot en présence d'un segment d'obstacle

Dans ce cas, pour que le robot atteigne le point géométrique final situé sur la droite Δ, il doit procéder en deux étapes :

- il doit d'abord atteindre la droite Δ parallèle au segment et passant par la configuration à atteindre, en utilisant la méthode qui utilise la commande sur la vitesse angulaire défini précédemment. Cette méthode ne permet au robot que d'atteindre la droite Δ puisqu'elle ne permet pas d'avoir plus d'information sur la configuration finale du robot.
- En connaissant la position relative du but par rapport à la source, on peut choisir la direction que le robot doit prendre sur la droite Δ pour se diriger vers la configuration but. On utilise à ce moment une commande pour suivre parallèlement le segment d'obstacle.

La commande qui permet au robot de longer un segment d'obstacle est définie par :

$$\dot{\theta} = k_l(-\theta_d + \theta r_s + signe \times \frac{\pi}{2})$$

où $\theta \mathbf{r_s}$ représente l'orientation de la normale du segment d'obstacle par rapport au référentiel global que le robot doit longer, **signe** prend la valeur +1 ou -1 selon le sens entrepris par le robot pour parcourir la droite Δ. $(\theta r_s + signe \times \frac{\pi}{2})$ correspond à l'orientation que doit prendre le robot à partir de la configuration P_Δ située sur la droite Δ pour se diriger dans le sens de la configuration but. $\mathbf{k_l}$ est un coefficient donné.

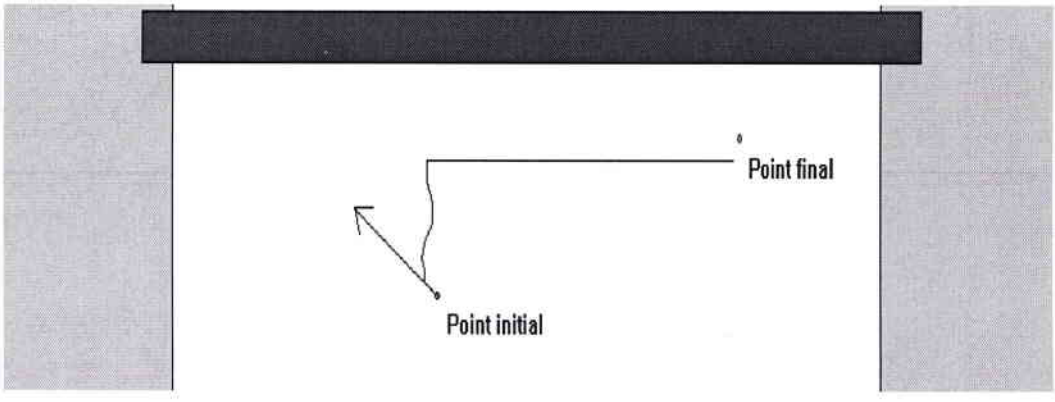


Figure 3.7

Résultat de commande sur l'orientation dans le cas d'un segment infini
Le tiré au point initial indique la direction (orientation) initiale du robot.

Cette procédure qui permet de longer un segment d'obstacle est importante dans le cas d'un déplacement vers une autre région. Dans notre cas, aucune information n'est disponible sur les relations de proximités géométriques entre la position du robot et le point à atteindre.

3.2.4.2 - Cas de deux segments d'obstacles parallèles

Les mesures des capteurs sont les mêmes sur toute configuration située sur une droite parallèle aux deux segments d'obstacles. On est dans la même situation que précédemment. Le robot tend d'abord vers une droite Δ, où se trouve la configuration but. La connaissance de la position de la configuration but par rapport à la configuration source lui permet de choisir le sens dans lequel il doit longer les deux segments d'obstacles le long de la droite Δ.

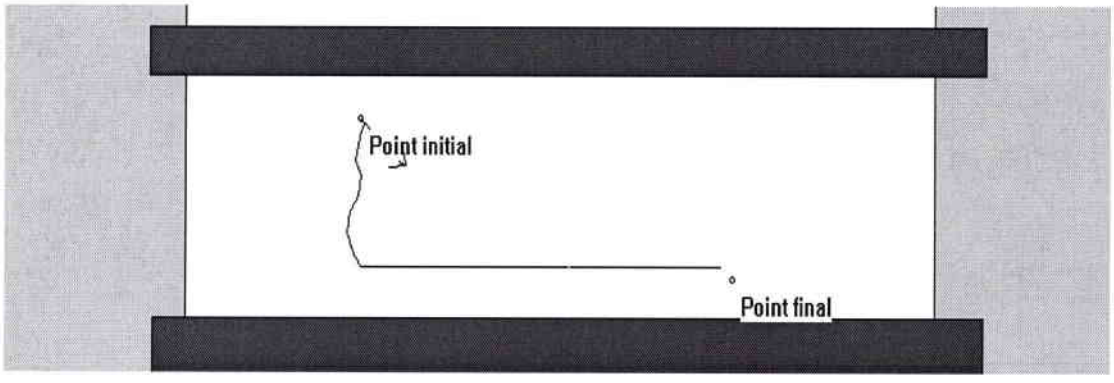


Figure 3.8
Exemple de commande sur l'orientation du robot dans un couloir
(deux segments d'obstacles parallèles).

La commande sur l'orientation à appliquer sur le robot pour longer les deux segments d'obstacles parallèles est identique à celle appliquée pour longer le segment d'obstacle dans le cas précédent.

3.2.4.3 - Cas de plusieurs segments non parallèles

Dans le cas général, le robot tend directement vers la configuration finale. Le coefficient k utilisé dans la commande sert de paramètre pour réguler le déplacement angulaire du robot en fonction de l'erreur e qui permet de comparer les mesures des capteurs en temps réel avec celles attendues à la configuration finale définies dans le modèle.

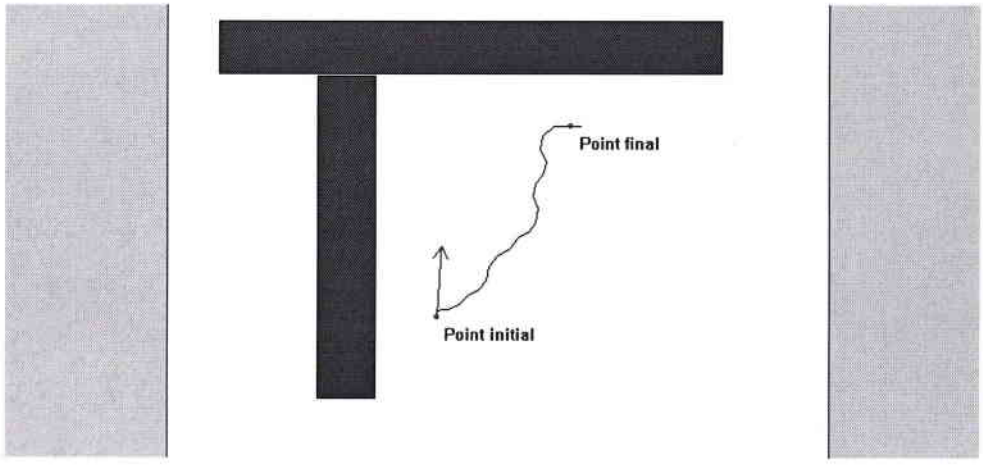


Figure 3.9 a

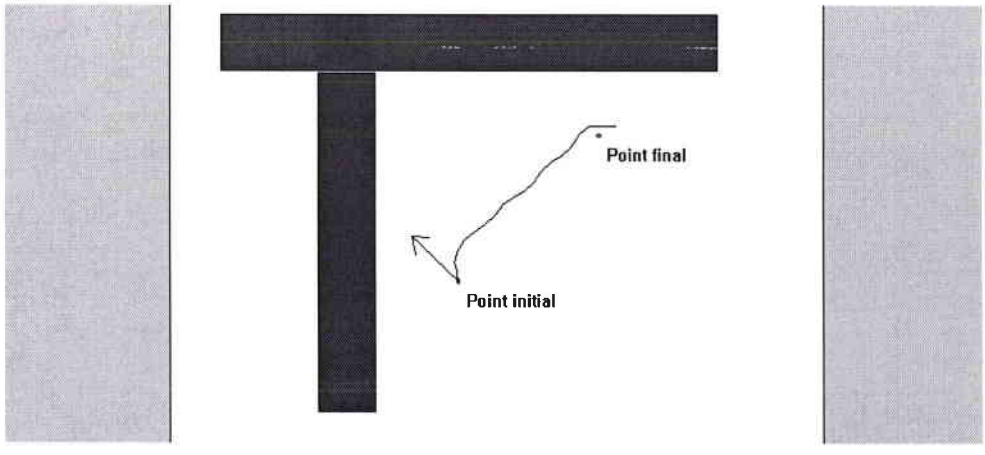


Figure 3.9 b

Résultat de commande sur l'orientation du robot dans le cas
de deux segments d'obstacle orthogonaux
pour k=0.2 (a) et k=0.5 (b)

3.3 - Changement de commandes

Lorsqu'on veut passer de la commande sur la variation angulaire à la commande qui permet le suivi d'un segment d'obstacle, on peut avoir un chemin dont le rayon de courbure important est difficile à suivre en temps réel compte tenu de la dynamique du véhicule. Ce rayon de courbure doit être inférieur à la vitesse angulaire maximale autorisée $\dot{\theta}_{max}$ entre deux configurations successives du robot. Ce faisant, seules les rotations inférieures à $\dot{\theta}_{max}$ sont autorisées durant la planification.

La commande qui permet de longer un mur a été définie précédemment par :

$$\dot{\theta} = k_c(-\theta_d + \theta r_s + signe \times \frac{\pi}{2})$$

où $\mathbf{k_c}$ est un coefficient donné.

Si $|\dot{\theta}| > \dot{\theta}_{max}$ pendant la phase de transition, alors la commande $\dot{\theta}$ pendant ce temps aura la valeur :

$$\dot{\theta} = \frac{\theta r_c + signe \times \frac{\pi}{2} - \theta_d}{\left|\theta r_c + signe \times \frac{\pi}{2} - \theta_d\right|} \dot{\theta}_{max}$$

où $\dot{\theta}$ prend la valeur de $\dot{\theta}_{max}$ avec comme signe celui de $(-\theta_d+\theta r_s+ signe \times \frac{\pi}{2})$.

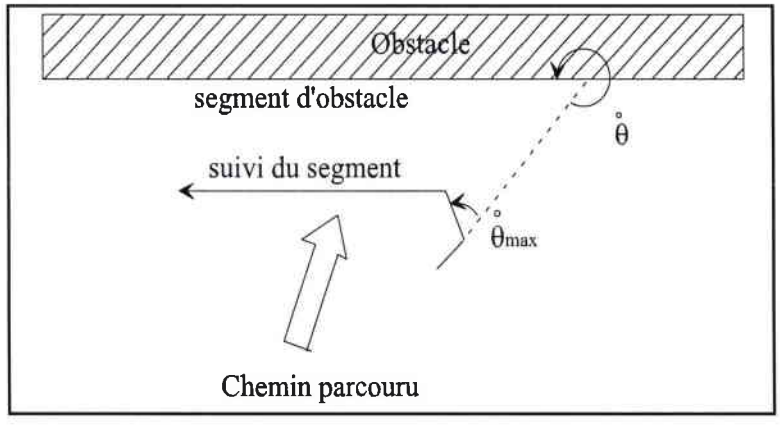


Figure 3.10 : Cas discret où $|\dot{\theta}| > \dot{\theta}_{max}$. Changement progressive d'orientation avant que le robot ne longe le segment d'obstacle

3.4 - Passage d'une région à une autre

La commande qu'on a défini précédemment permet au robot de se déplacer d'une région vers une autre région adjacente. Comme le robot est amené dans le cas général à traverser plusieurs régions avant d'atteindre le but, il serait intéressant de connaître le moment où le robot entre dans une nouvelle région afin de définir la nouvelle configuration à prendre en compte.

3.4.1 - Conditions de passage

Pour que le robot passe d'une région à une autre, on a deux cas :

- le robot s'éloigne d'un segment d'obstacle jusqu'à ce que le segment ne soit plus détectable par les capteurs du robot pour une valeur maximale des mesures.
- le robot s'approche d'un segment d'obstacle jusqu'à ce que les capteurs du robot le détecte.

On a vu dans le chapitre précédent que le passage entre deux régions était du à un changement d'information entre ces deux régions. Cela correspond à la détection ou la perte de l'information d'un segment. Cette information est constituée par l'orientation du capteur qui a détecté le segment d'obstacle pour laquelle la distance mesurée est la plus courte, dans une seule des deux régions.

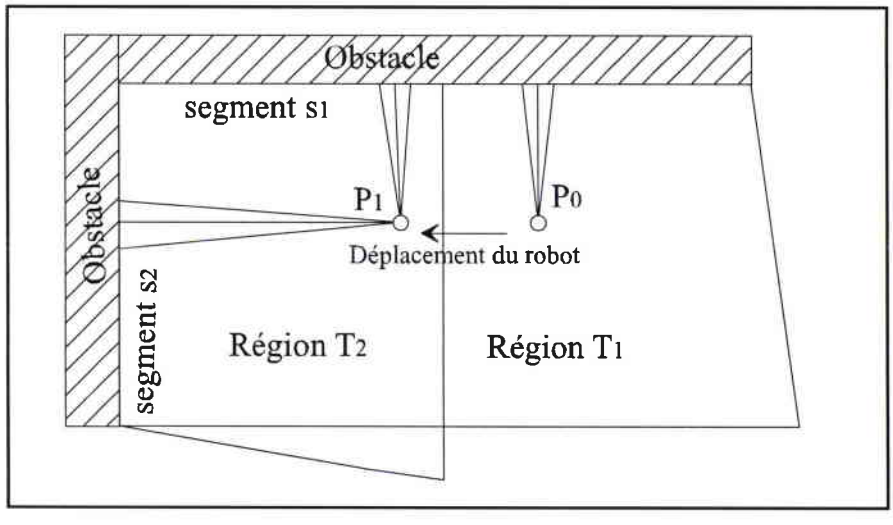


Figure 3.11 : Représentation du passage du robot
entre deux régions qui entraîne le changement d'information

En représentant les mesures des capteurs obtenues dans les régions T_1 et T_2 en fonction de l'orientation à laquelle elles sont effectuées dans le référentiel global, on constate que le changement d'information s'effectue autour de l'orientation θ_{s2}. Le segment s_2 est alors détecté par le robot dans T_2. (voir figure 3.12)

Pour savoir si le robot est passé dans la région T_2, on vérifie si le signal constitué par les mesures des capteurs (segment s_2) apparaît. De même, si le robot est amené à faire le passage inverse de T_2 vers T_1, on peut connaître l'instant où le robot passe dans T_1 lorsque le signal autour de θ_{s2} disparaît.

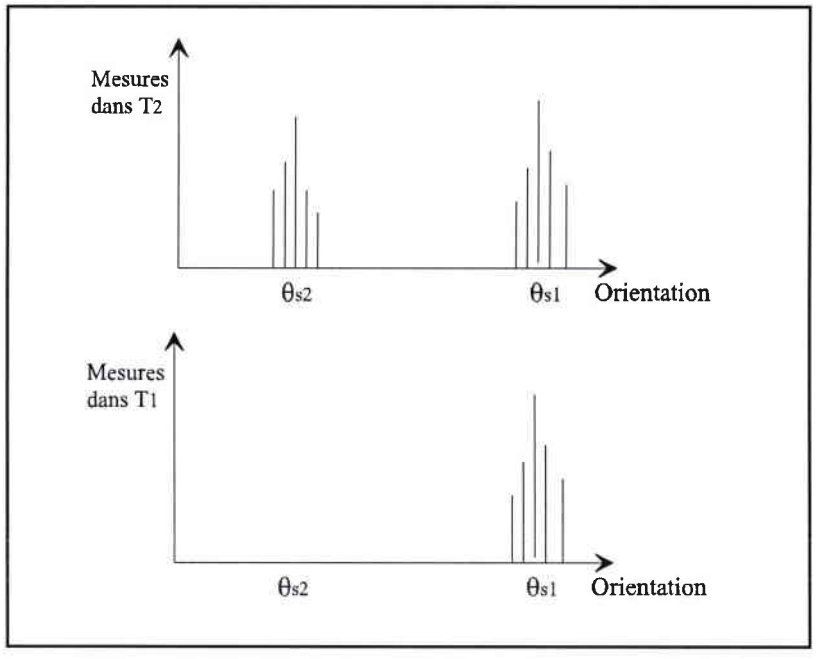


Figure 3.12 : Variation d'information autour de θ_{s2}
entre les deux régions T_1 et T_2

3.4.2 - Algorithme

Soit **PlusInfo** une valeur booléenne qui prend la valeur vraie si la région dans laquelle se situe le robot contient une information supplémentaire (ou dans un autre sens détecte plus de segments d'obstacles) par rapport à la région suivante sur le chemin vers le but. Dans le cas contraire, **PlusInfo** prend la valeur faux. **PlusInfo** est déterminée en comparant le nombre d'informations entre deux régions adjacentes faisant partie du chemin optimal. Les informations obtenues dans chaque région du chemin sont celles qui ont été calculées lors de la modélisation de l'environnement et qui sont contenues dans les noeuds du graphe. Le

chemin optimal est calculé par l'intermédiaire de l'algorithme de Dijkstra (voir paragraphe précédent).

M_d est une valeur qui représente une mesure (distance robot-obstacle la plus courte) des capteurs dans le cas réel, lorsqu'il y a apparition d'un signal ou d'une information sur un nouveau segment d'obstacle. $M_d = 0$ dans le cas où un signal ou une information sur un segment disparaît.

L'algorithme pour la condition de passage dans une nouvelle région est le suivant :

```
Initialisation i=1
Tant que le robot n'a pas atteint la dernière région du chemin
Début
    Si i≠1 Passage à la configuration suivante du robot
        Si (PlusInfo est vrai et (Md == 0)) ou (PlusInfo est faux et (Md > 0))
            alors le robot est passé dans une nouvelle région
        Sinon le robot est toujours dans la même région
    Incrémentation i=i+1
Fin
```

3.4.3 - Calcul de l'erreur pour le passage d'une région à une autre

On se situe dans le cas où le point à atteindre est dans une région et la position du robot dans une région adjacente. Pour que le robot puisse passer convenablement de la région où il est situé jusqu'à la région suivante où se trouve le point à atteindre, il faut que l'erreur e entre le point à atteindre (configuration du modèle) et celui du point actuel (configuration réelle) soit réalisée uniquement sur les mêmes segments d'obstacles. On ne prend pas en compte la mesure sur un segment d'obstacle qui est obtenue dans une seule des deux régions.

Reprenons l'exemple de la figure 3.11. Le robot obtient des mesures sur le segment s_1 dans les deux régions T_1 et T_2 où doit s'effectuer le passage, par contre il n'obtient des mesures sur le segment s_2 uniquement dans la région T_2. Ainsi l'erreur e à minimiser n'est considérée que sur les mesures réalisées sur le segment s_1.

On prend en compte, dés que le robot est passé dans la région T_2 où est situé le point à atteindre, les mesures réalisées sur le segment d'obstacle s_2 dans le calcul de l'erreur e.

Il est possible, mais dans des cas rares, que le nombre de mesures effectuées sur un segment d'obstacle varie d'une configuration à une autre et peut influencer la trajectoire du robot. Pour cette raison, on prend la moyenne des mesures effectuées sur un même segment d'obstacle dans le calcul de l'erreur. On aura ainsi toujours un même nombre de mesures pour un segment d'obstacle.

Dans l'exemple de la figure 3.11 on a les valeurs suivantes de e lors du passage de T_1 à T_2:

- Avant le passage à la région T_2

$$e = \text{moy}\left(\int_{\theta=\theta s1-\theta s}^{\theta=\theta s1+\theta s} d(\theta)\delta\theta\right)\text{-moy}\left(\int_{\theta=\theta s1-\theta s}^{\theta=\theta s1+\theta s} d'(\theta-\theta_d)\delta\theta\right)$$

- Après le passage à la région T_2

$$e = e(\text{avant le passage}) + \text{moy}\left(\int_{\theta=\theta s2-\theta s}^{\theta=\theta s2+\theta s} d(\theta)\delta\theta\right)\text{-moy}\left(\int_{\theta=\theta s2-\theta s}^{\theta=\theta s2+\theta s} d'(\theta-\theta_d)\delta\theta\right)$$

avec θs_1 et θs_2 respectivement les orientations correspondant aux mesures effectuées sur les segments s_1 et s_2 aux distances les plus courtes et θs l'angle de réflexion maximal du faisceau d'un capteur.

3.5 - Utilisation de configurations de référence.

Dans chaque région faisant partie du chemin du robot, on choisit une configuration de référence qui permet au robot de tendre vers cette configuration tant qu'il se situe dans la région précédente. La configuration de référence constitue un but potentiel pour le robot tant qu'il n'est pas entré dans la région où se situe cette configuration. L'intérêt principal de l'utilisation de configurations de référence est de permettre de comparer les mesures des capteurs en temps réel avec des mesures spécifiques et idéales obtenues sur les configurations de référence. Ces mesures constituent les informations fournies au système par le modèle de l'environnement. Une configuration de référence est choisie dans une région selon une procédure définie ci-après.

3.5.1 - Choix des configurations de référence.

Comme on a vu au chapitre précédent, pour accéder aux régions créées à partir des informations capteurs, on utilise un handle. On pourrait y accéder aussi, si on a connaissance de tous les points représentant les sommets de ces régions. A partir des sommets d'une de ces régions, on pourrait trouver une configuration de référence au centre de cette région. Ce n'est pas le cas, puisqu'on ne connaît a priori que les coordonnées des sommets des régions créées à partir des informations capteurs qui sont en même temps des sommets d'obstacle. Il n'existe pas une fonction sous *Windows* qui permet de récupérer les coordonnées des autres sommets de ces régions. Par contre on dispose d'une fonction permettant de définir l'appartenance d'un point à une région. Des points de référence sont choisi aléatoirement dans chacune des régions faisant partie du chemin.

On parcourt toute la liste contenant les régions par lesquelles le robot devra passer, et pour chacune de ces régions, on vérifie si un point choisi aléatoirement par l'ordinateur fait partie de ces régions.

L'algorithme pour trouver un point de référence est le suivant :

```
Initialisation i=1
Tant que le robot n'a pas atteint la dernière région du chemin
Début
        Si i≠1 Passage à la région suivante du chemin
                Tant qu'une configuration aléatoire ∉ à la i^ème région du chemin
                        Choisir une autre configuration
        Incrémentation i = i+1
Fin
```

Pour une région assez grande, le temps qu'il faut pour trouver un point de référence est négligeable, alors que pour une région très petite, il faut parfois beaucoup plus de temps. On a donc intérêt à enlever les régions de l'environnement de dimensions très petites, en plus du fait que le robot aura des difficultés à traverser ces régions. La méthode qui permet de centrer

un point de référence dans une région va aussi nous permettre d'identifier les régions indésirables pour qu'on puisse les supprimer par la suite.

3.5.2 - Centrage des configurations de références

L'intérêt de pouvoir centrer un point de référence à l'intérieur d'une région, permet au robot de pénétrer suffisamment dans la région le temps qu'il puisse trouver la bonne direction pour se diriger vers la région suivante sur le chemin vers le but final.

Le fait de pouvoir centrer un point de référence permet d'éviter deux cas :

- Si le point de référence choisi aléatoirement est situé très proche de la frontière d'une région ne faisant pas partie du chemin du robot, il se peut que le robot y pénètre. Il doit à ce moment retourner dans la région dans laquelle se trouve le point de référence. Dans ce cas le chemin devient plus long.
- Si le point de référence choisi aléatoirement est situé très proche d'un segment d'obstacle, le robot risque de frôler ou heurter le segment d'obstacle, s'il doit s'approcher suffisamment près du point de référence pour pouvoir ensuite passer à la région suivante.

3.5.2.1 - Principe

Le principe consiste à trouver une droite qui passe par un point choisi de manière aléatoire, pour une direction ou pente quelconque. On se déplace sur cette droite le long des deux demi-droites situées sur chaque côté du point choisi aléatoirement selon un pas donné en comptant le nombre de pas de chaque côté du point qu'il faut pour sortir de la région. On se déplace alors sur la droite du côté où le nombre de pas est le plus important, jusqu'à ce qu'on trouve un point qui soit équidistant des deux côtés par rapport aux limites de la région dans laquelle est située le point. A ce moment on aura trouvé un point centré.

Sur la figure suivante, on a d_1 et d_2 les distances qu'il faut, à partir du point P_a situé dans la région A, pour sortir de la région A des deux côtés du point P_a situé sur la droite d. d_1=7pas et d_2=2pas. Comme $d_1>d_2$, pour trouver un point centré sur la droite d, il faut se déplacer le long de cette droite, du côté où on a la distance d_1 vers la sortie de la région A, jusqu'à ce que d_1 soit à un pas près égal à d_2.

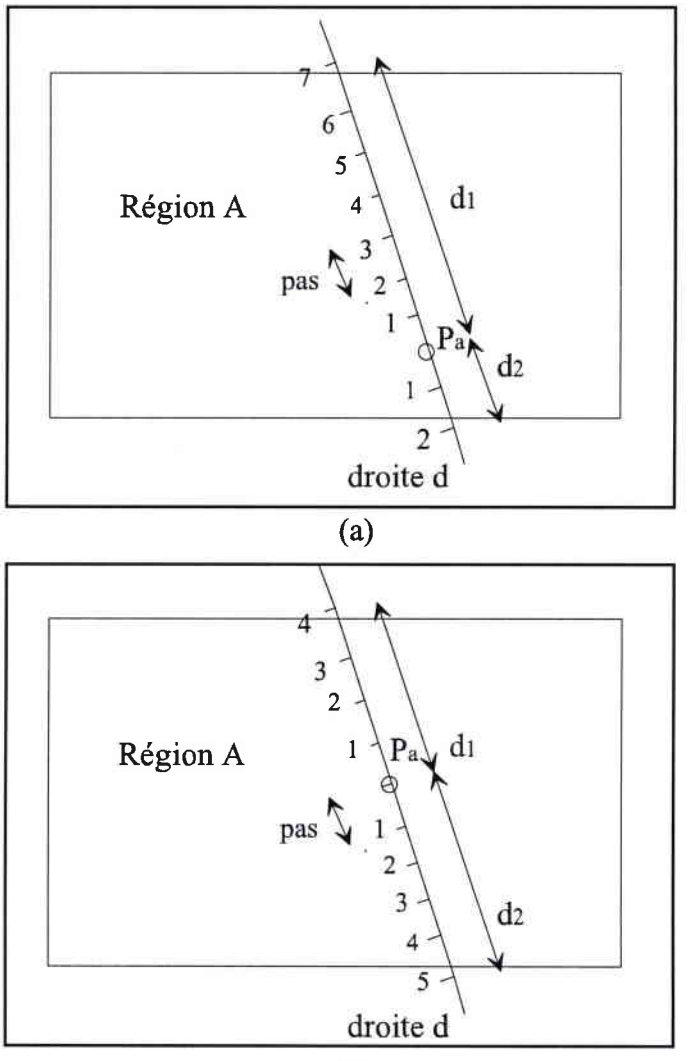


Figure 3.13
Procédure pour centrer une configuration (b)
choisie aléatoirement (a), à l'intérieur d'une région.

L'orientation de la première droite qui permet de centrer un point, est choisie de manière quelconque. Dès qu'un point centré c_1 est trouvé à travers une droite Δ_1 entre deux limites de la région dans laquelle est située le point, on cherche un autre point centré c_2 à travers une autre droite Δ_2 passant par c_1 et orientée de $\frac{\pi}{n_d}$ par rapport à Δ_1. n_d est le nombre total de droites utilisées pour trouver un point de référence centré dans une région du chemin.

On peut se servir de cette procédure pour trouver les régions de petites dimensions indésirables qui ont des dimensions plus petites que celles du robot. Ces régions ont au moins

deux limites à des distances très proches, c'est à dire plus petit qu'une valeur admissible V_a qui peut être en relation avec la dimension du robot. Il faut vérifier si la somme :

$$s = d_1 + d_2 < V_a$$

d_1 et d_2 étant les distances respectives entre une configuration centrée avec les deux limites de la région où elle se situe.

3.5.2.2 - Algorithme

Soit a_i la pente de la droite d_i d'équation dans un repère cartésien $y = a_i x + b$, qui passe par une configuration choisie aléatoirement $p_a(x_a, y_a)$. p_{d1} et p_{d2} sont respectivement les deux configurations qu'on obtient à chaque déplacement d'un pas dans les deux directions opposées de la droite d_i à partir du point p_a. $p_c(x_c, y_c)$ est le point de référence recherché. R_a est la région dans laquelle est situé le point p_a. i, k_1 et k_2 sont des indices.

Initialisation i=0
Tant que $i < n_d$
Début
    Initialisation k_1=0 et k_2=0, $p_{d1} = p_{d2} = p_a$
    Tant que $p_{d1} \in R_a$
        $p_{d1}.x = x_a - (k_1 \times pas)$
        $p_{d1}.y = y_a - (k_1 \times a_i \times pas)$
        Incrémentation de k_1
    Tant que $p_{d2} \in R_a$
        $p_{d2}.x = x_a + (k_2 \times pas)$
        $p_{d2}.y = y_a + (k_2 \times a_i \times pas)$
        Incrémentation de k_2
    $x_c = x_a + (k_2 - k_1) \times pas$
    $y_c = y_a + (k_2 - k_1) \times a_i \times pas$
    $p_a = p_c$
Fin

3.6 - Evitement d'obstacle en simulation

Lorsque le robot se rapproche trop d'un segment d'obstacle et qu'il se retrouve à une distance minimale d_{min}, il risque d'entrer en collision avec cet obstacle. On propose alors de changer l'orientation du robot dès qu'il se trouve à une distance plus petit que d_{min} du segment d'obstacle, pour qu'il s'en éloigne et éviter la collision. On augmente la vitesse (variation) angulaire du robot au maximum, c'est à dire pour $\Delta\theta = \Delta\theta_{max}$,pour qu'il puisse changer d'orientation et on diminue la vitesse linéaire jusqu'à une vitesse minimale $v_e = v_{emin}$. Dès que le robot s'est suffisamment éloigné du segment d'obstacle et qu'il se retrouve à une distance supérieure à d_{min}, la vitesse angulaire et la vitesse linéaire seront à nouveau définies en fonction de l'erreur e entre les mesures des capteurs.

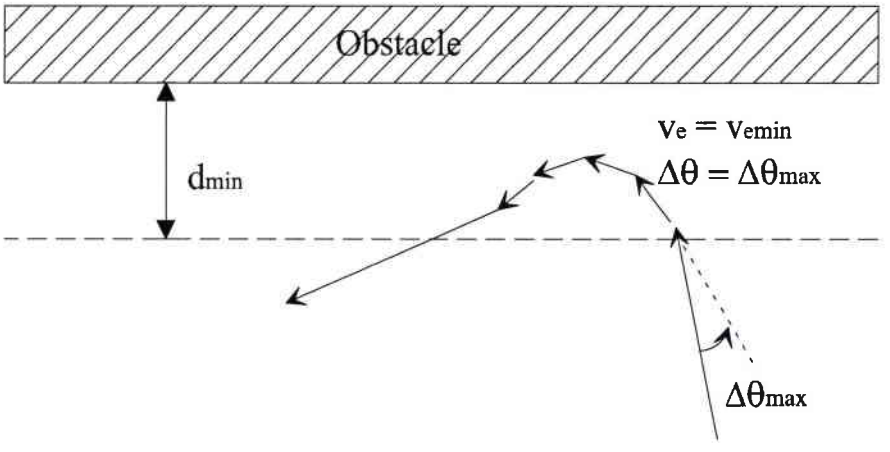


Figure 3.14 Représentation d'un exemple de chemin
parcouru par un robot pour éviter la collision avec les obstacles

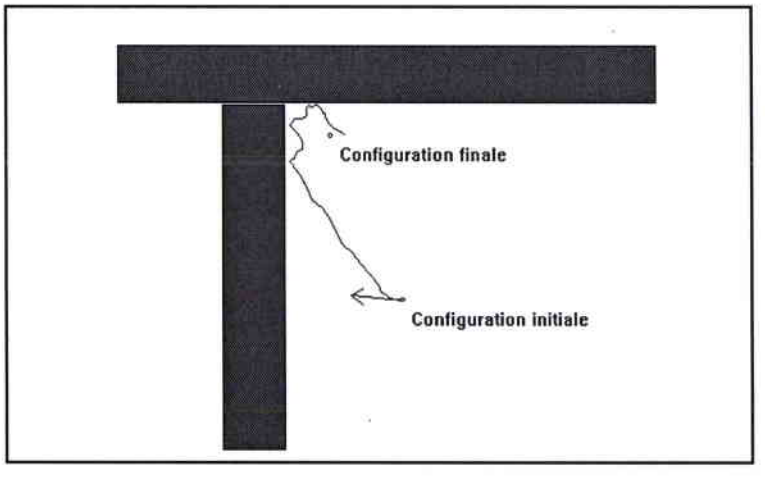


Figure 3.15 Exemple de chemin du robot
avec évitement d'obstacle

3.7 - Commande sur plusieurs régions

Pour que le robot passe à travers toutes les régions sur le chemin du but, on utilise une combinaison de la commande sur le déplacement angulaire du robot et la commande pour suivre parallèlement un segment d'obstacle.

Par défaut, on utilise la commande sur le déplacement angulaire. On utilise la commande de suivi d'un segment d'obstacle à chaque fois que l'écart e de mesures entre celles du point à atteindre dans le modèle (configuration de référence) et celles obtenues en temps réel est plus petit qu'une valeur donnée e_{min}. A ce moment, les informations obtenues par le robot sur les segments d'obstacles sont quasiment identiques d'une configuration à une autre, la commande sur le déplacement angulaire qui permet de minimiser l'erreur e ne peut plus être utilisée puisque e est trop petit. On se sert dans cette situation de la commande de suivi d'un segment d'obstacle pour permettre au robot d'atteindre la région suivante. Dès que le robot a atteint une nouvelle région, on réutilise à nouveau la commande sur le déplacement angulaire pour $e>e_{min}$. (voir exemple de résultats figure 3.16)

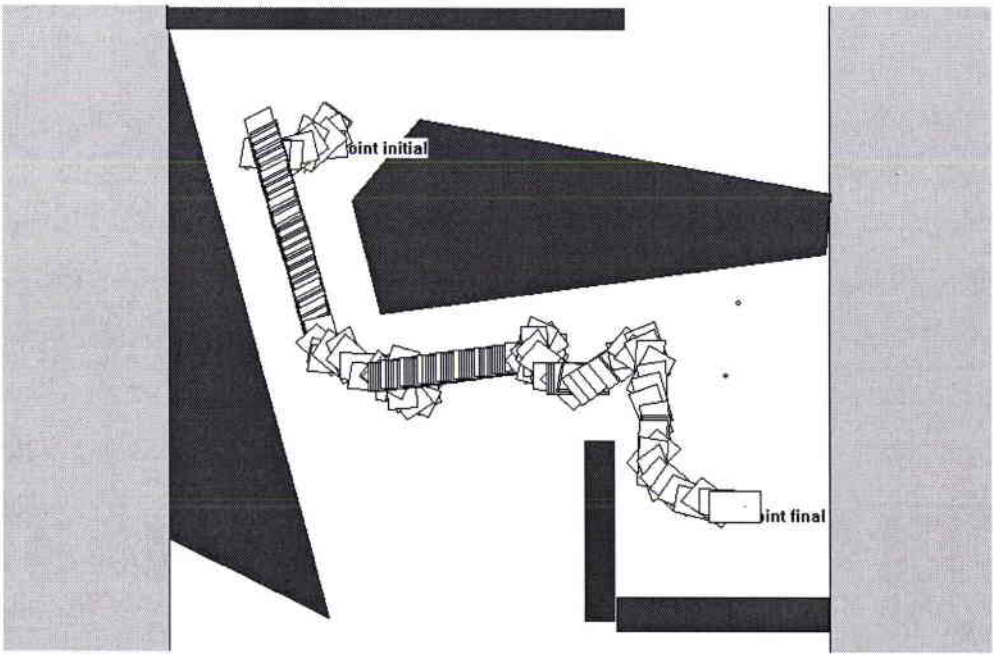


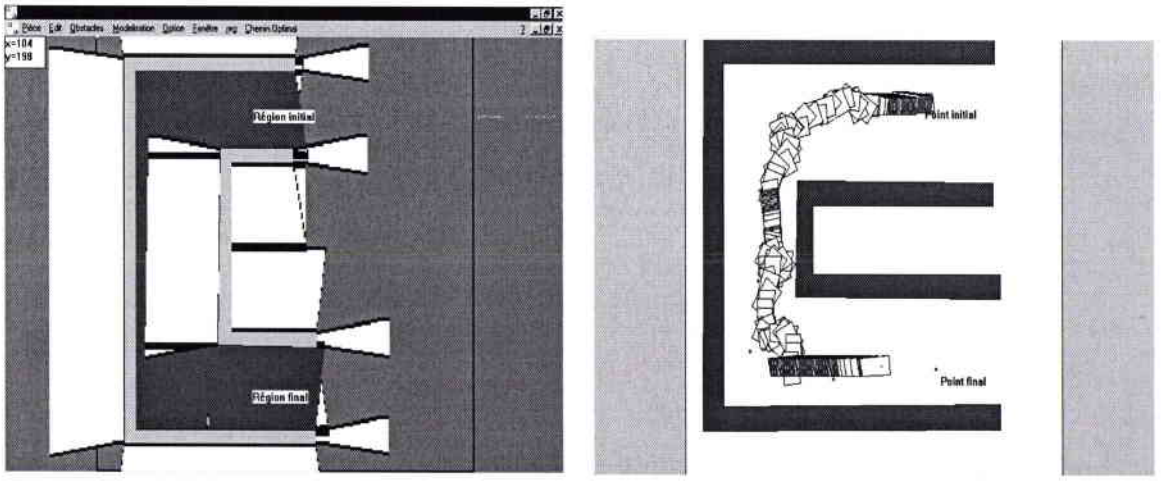
Figure 3.16 Exemple de résultats de chemin obtenue
à travers plusieurs régions du modèle d'un point initial à un point final

L'algorithme pour commander la trajectoire du robot est donc le suivant :

Tant que le robot n'a pas atteint la dernière région du chemin
{
Si le robot entre dans une nouvelle région
Commandes sur la vitesse angulaire et linéaire
Sinon si $e < e_{min}$
Commande pour le suivi d'un segment d'obstacle
}

3.8 - Incertitudes

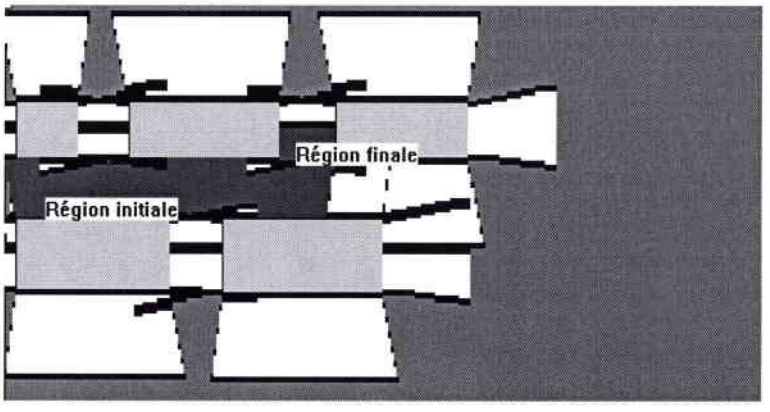
Dans cette partie, on veut vérifier la robustesse de la planification malgré les incertitudes issues du modèle de l'environnement et des mesures des capteurs. Ces incertitudes sont déterminées en effectuant des modifications sur l'environnement réel par rapport au modèle. Ceci est réalisé en augmentant ou en rétrécissant la longueur des segments d'obstacle, en rendant un couloir plus large ou plus étroit. Nous vérifions que malgré ces modifications, le robot arrive à trouver le bon chemin au but.



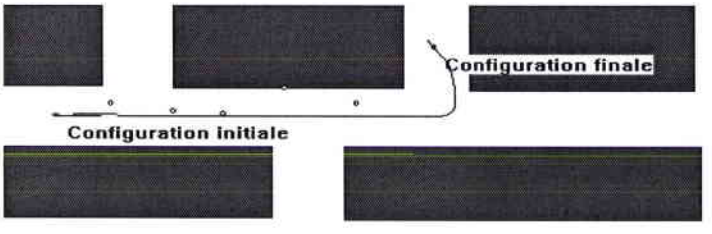
(a) (b)

Figure 3.17a et b : Représentation du modèle (a) et de la planification de trajectoire tenant compte des incertitudes (b).

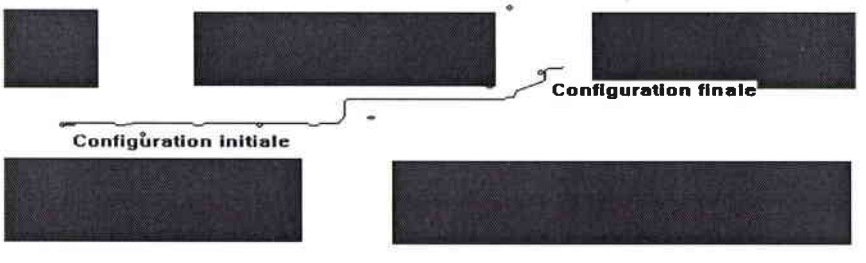
On a aussi un autre exemple sur le passage de portes qui montre la robustesse de la commande par rapport aux erreurs du modèle, lorsque dans le cas réel, le couloir est plus long que celui défini dans le modèle de l'environnement. On voit bien que le robot en temps réel passe par la deuxième porte à gauche du couloir comme indiqué par le chemin en couleur grise dans le modèle.



(a)



(b)



(c)

Figure 3.18

(a) Planification de trajectoire du robot lors de passage de portes en gris foncé.
(b) et (c) Exemples d'exécution de configurations réelles.

3.9 - Capteurs statiques utilisés

3.9.1 - Utilisation de capteurs statiques situés dans l'axe du centre du robot

Jusqu'à présent, toute l'étude a porté sur l'utilisation d'un capteur unique en rotation qui permettait de balayer l'environnement. Dans le cas de l'utilisation sur un fauteuil, qui est notre objectif final, nous utilisons un ensemble de capteurs à ultrasons statiques disposés tout autour du mobile.

Dans ce paragraphe, nous allons analyser le comportement du robot en fonction du nombres de capteurs, c'est à dire en fonction de l'espace perçu. Chaque capteur a un cône d'émission de 24° et est disposé tout autour du robot selon un intervalle angulaire régulier θ. L'axe de chaque capteur se coupe en un même point P_c afin que les mesures soient les plus proches possibles de celles effectuées avec le capteur rotatif.

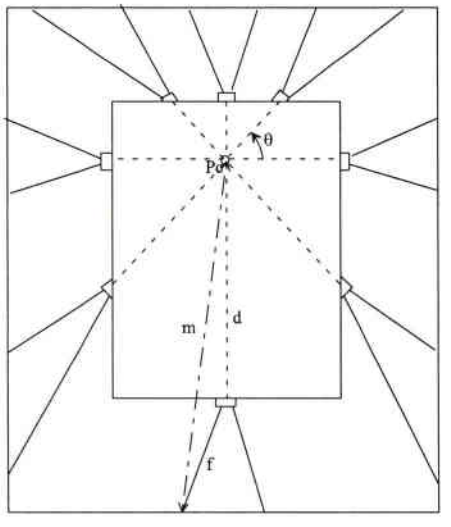


Figure 3.19 : Exemple de capteurs statiques disposés
tout autour du robot selon un intervalle angulaire régulier

Pour chaque capteur fixe, la mesure à l'intérieur du cône d'émission correspond à la distance la plus courte par rapport à un segment d'obstacle. Afin que la mesure de chaque capteur statique soit calculée par rapport à un même repère P_c, comme dans le cas d'un capteur tournant, on ajoute à chaque mesure obtenue par rapport à un capteur statique, la

distance qu'il y a entre celui-ci et le centre du robot. On réalise une approximation de la mesure par rapport à P_c en considérant m = f + d. (voir figure 3.19).

Plus le nombre de capteurs sera élevé, plus l'espace autour du robot sera balayé. Dans ce cas, l'observation de l'environnement se rapproche de celle qu'on obtient avec la configuration du capteur rotatif, jusqu'à ce que l'espace autour du robot soit balayé en entier par les capteurs utilisés en nombre suffisant.

Afin de vérifier la limite de validité de notre approche, nous avons diminué le nombre de capteurs statiques. En réduisant jusqu'à 20 le nombre de ces capteurs fixes utilisés, l'ensemble des informations sur les segments d'obstacles nécessaires à l'obtention d'une bonne planification de trajectoire est acquis, bien que les capteurs ne couvrent plus l'ensemble de l'environnement. La trajectoire du robot est similaire au cas où on utilise un capteur rotatif.

En diminuant le nombre de capteurs fixes utilisés en simulation à 10, la trajectoire du robot vers le but devient plus longue car l'erreur e entre les mesures augmente, due aux informations de plus en plus incomplètes sur l'environnement. A partir de 10 capteurs, le nombre de mesures insuffisants des capteurs ne permet plus au robot de trouver un chemin vers le but. L'erreur e s'annule avant de pouvoir atteindre le but à cause d'absences complètes de mesures pour certaines orientations du robot.

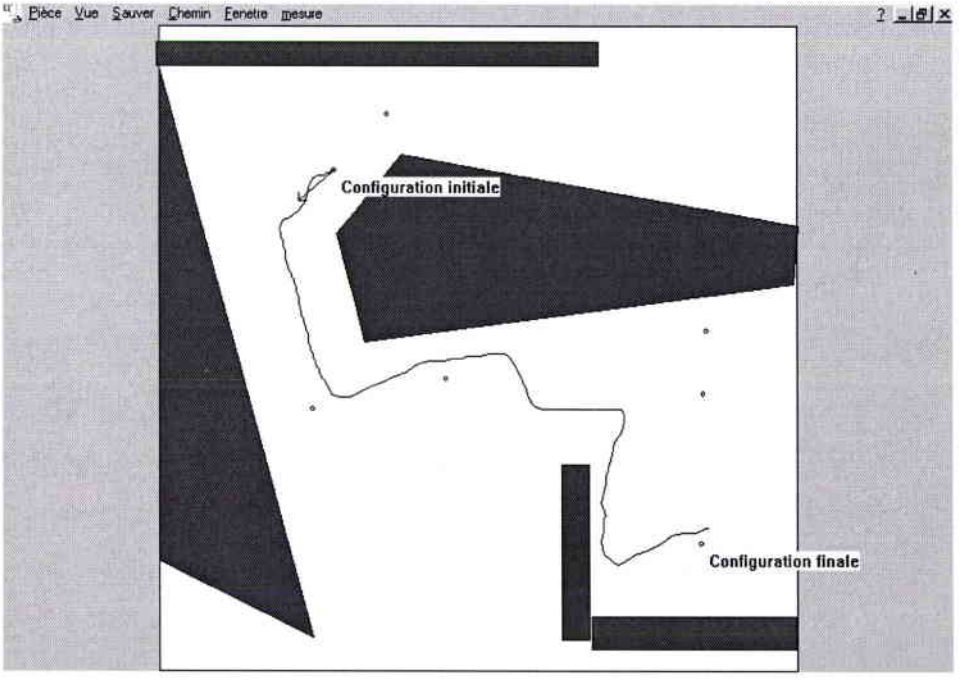


Figure 3.20 : Exemple de planification de trajectoire
en utilisant 30 capteurs statiques.

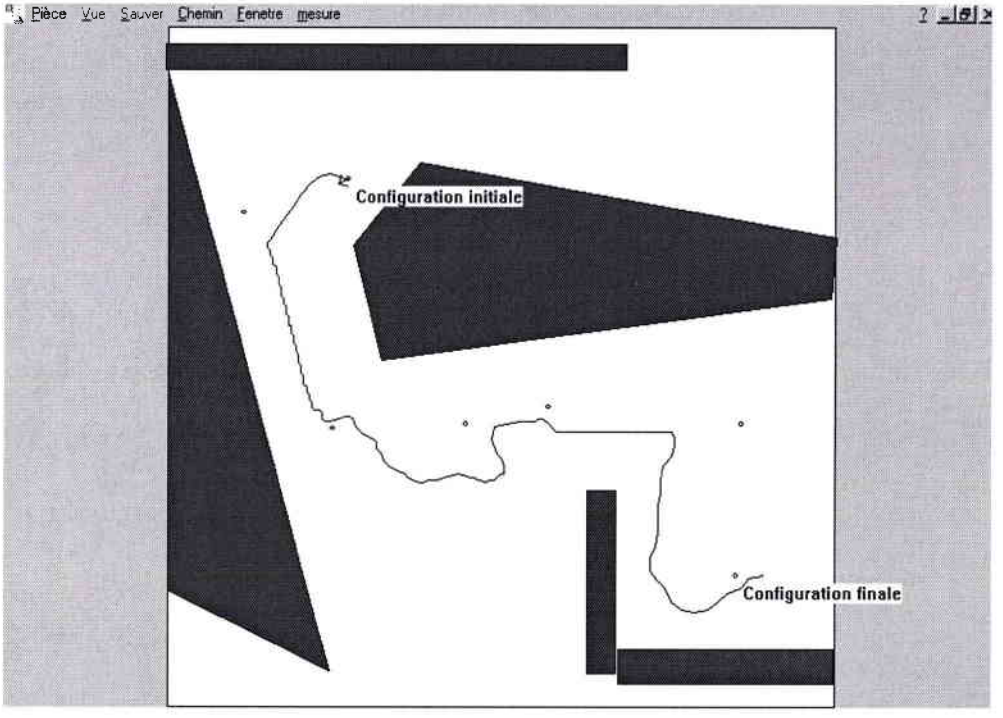


Figure 3.21 : Exemple de planification de trajectoire
en utilisant 25 capteurs statiques.

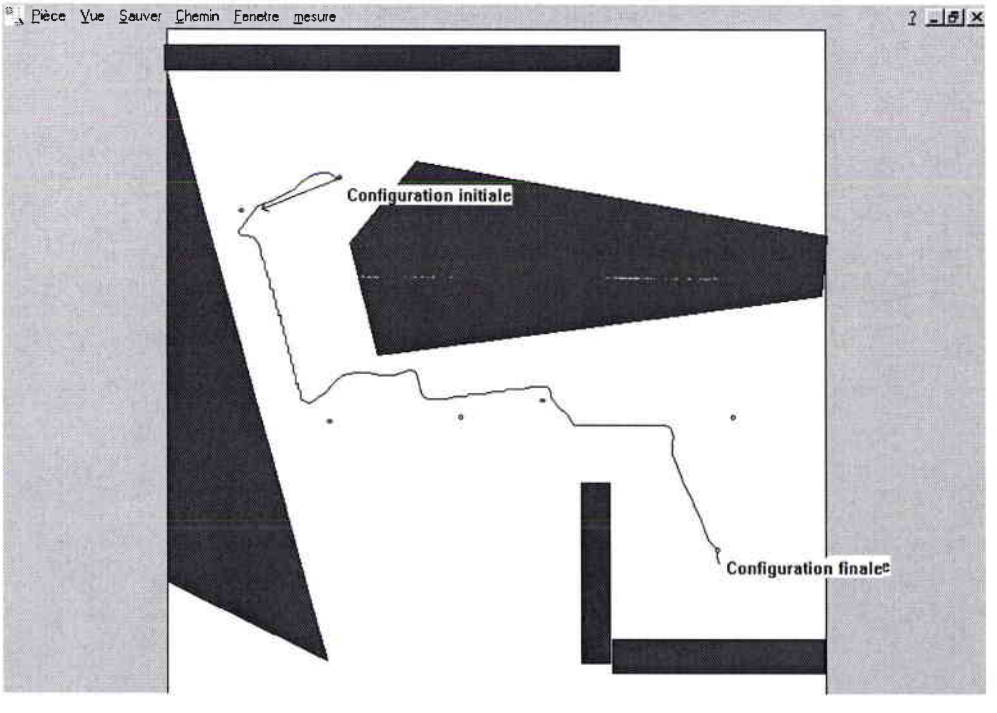


Figure 3.22 : Exemple de planification de trajectoire
en utilisant 15 capteurs statiques.

3.9.2 - Utilisation des capteurs tels qu'ils sont disposés sur le fauteuil

Si l'on utilise les 16 capteurs fixes du fauteuil roulant de notre laboratoire tels qu'ils sont disposés (voir figure 3.23), on remarque que la plupart des capteurs, en dehors des capteurs 4 et 13, ne sont pas situés dans l'axe du point de repère P_c .

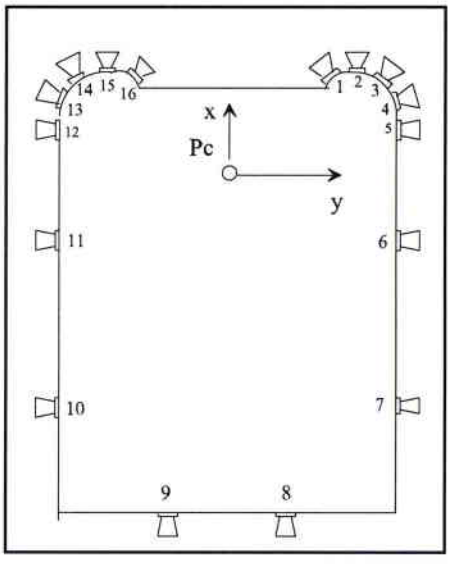


Figure 3.23 : Emplacements
des capteurs statiques sur fauteuil

Soit P_m un point sur lequel est réalisé la réflexion d'une onde émise et reçue par un capteur qui n'est pas situé dans l'axe du point P_c. La mesure est référencée par rapport au point P_c. (figure 3.24)

Pour qu'une mesure d'un capteur statique effectuée sur le segment au point P_m puisse être utilisée, il faut que ce segment puisse être détecté au point P_m par un capteur tournant situé au point P_c, en tenant compte de l'angle de réflexion minimum autorisé α_r (voir fig.2.10) pour une onde émise du capteur sur ce segment. Si l'onde émise à partir d'un capteur tournant installé au point P_c décrit un angle θ_f par rapport au segment inférieur à α_r, alors l'onde réfléchie au point P_m ne peut pas être reçue par le capteur tournant. Dans ce cas, la mesure du capteur statique sur le segment au point P_m ne peut pas être utilisée. Si les ondes émises d'un capteur statique et du capteur tournant situé au point P_c ont des orientations θ_c et θ_f par rapport à un segment qui sont très proches, alors la mesure du capteur statique est prise en compte.

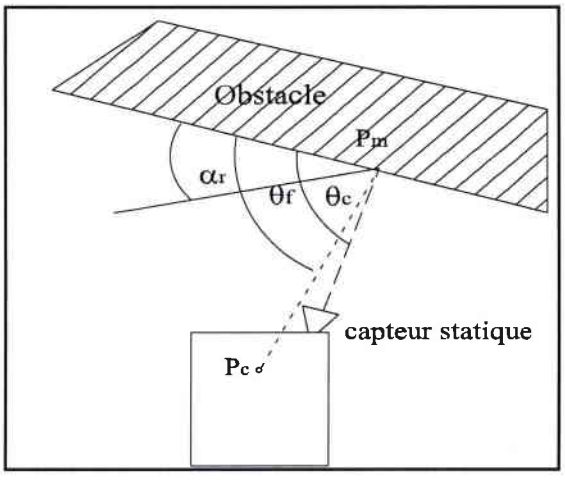
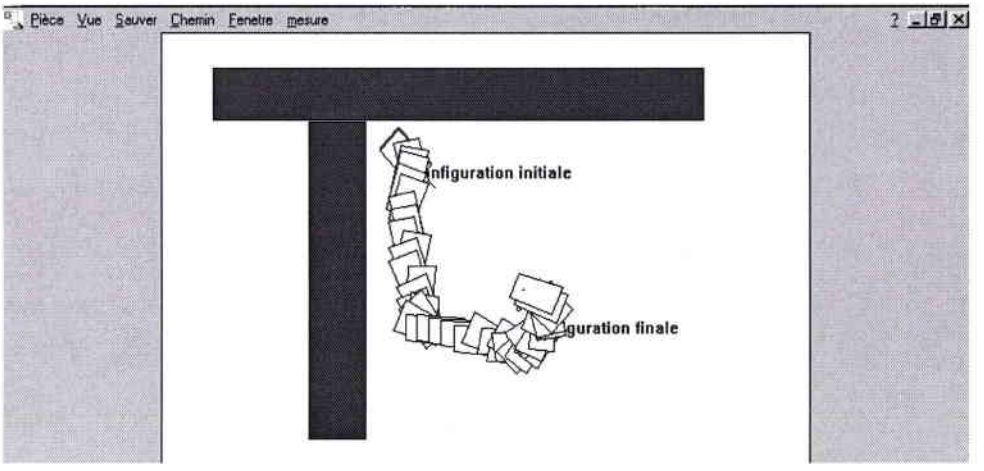


Figure 3.24 : Représentation des orientations des ondes émises d'un capteur statique et du capteur tournant (P_c) par rapport à un segment d'obstacle, en tenant compte de l'angle de réflexion α_r

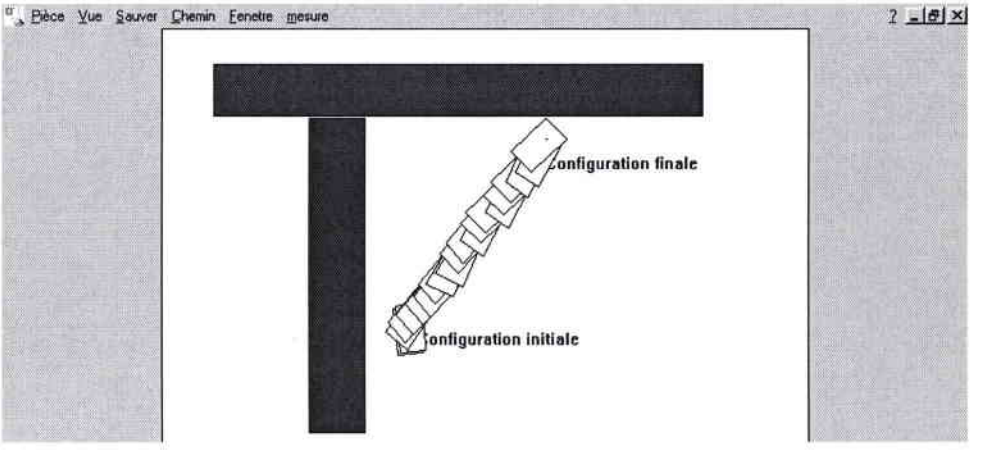
Si la mesure d'un capteur statique, qui n'est pas orienté dans l'axe du point P_c, est effectuée sur un segment d'obstacle qui se situe à une courte distance, les valeurs de θ_c et θ_f sont très différentes. Dans ce cas, la mesure du capteur statique n'est pas prise en compte. Par contre si la mesure est effectuée sur un segment d'obstacle beaucoup plus éloigné, de telle manière que les valeurs de θ_c et θ_f soient pratiquement identiques, alors cette mesure est utilisée. Les mesures qui sont effectués sur des segments d'obstacles, par des capteurs statiques qui ne se situent pas dans l'axe du point P_c, sont valables seulement si les segments sont suffisamment éloignés du robot. En simulation, nous n'avons pas tenu compte des mesures lorsque θ_c-θ_f>5°.

En utilisant les capteurs fixes du fauteuil, on obtient en simulation une trajectoire plus longue et moins orientée vers le but que celle obtenue avec un capteur rotatif ou avec des capteurs fixes (20 capteurs) réparties de manière régulière.

Le robot doit se servir essentiellement des capteurs situés à l'avant. Sur les figures 3.25a et b, on a deux cas différents de trajectoires pour le fauteuil. Sur la figure 3.25a, le robot tend moins rapidement vers la configuration but, car il se sert uniquement des 2 capteurs qui sont situés à l'arrière et des 2 capteurs situés sur la gauche pour obtenir des mesures sur les segments d'obstacles.



(a)



(b)

Figure 3.25 : Représentation de trajectoires
en utilisant les capteurs du fauteuil

Sur la figure 3.25b, le robot se dirige directement vers la configuration but. Il se sert d'un nombre plus élevé de capteurs que précédemment pour effectuer des mesures sur les segments d'obstacles. Des mesures sont réalisées avec des capteurs situés à l'avant (1 à 5) et des 2 capteurs situés sur la gauche du fauteuil.

3.10 - Conclusion

La commande que nous avons proposée, permet de suivre un chemin planifié, en tenant compte des incertitudes sur le modèle de l'environnement et sur la commande. Le robot est capable de trouver une trajectoire vers le but à atteindre malgré une modification locale de l'environnement par rapport au modèle.

D'après l'étude que nous avons menée sur la répartition des capteurs, nous pouvons déduire qu'il serait idéal que chacun des capteurs statiques soit situé sur un axe passant par le point origine P_c, afin que toutes les mesures obtenues à partir des capteurs statiques soient prises en compte. Comme la plupart des capteurs statiques sont situés à l'avant, ils couvrent entièrement la partie de l'environnement à l'avant du fauteuil. La partie à l'arrière n'est couverte que partiellement. Il serait intéressant d'ajouter à l'arrière du fauteuil des capteurs statiques orientés différemment par rapport à ceux déjà installés, afin de couvrir les parties inexplorées de l'environnement.

3.11 - Discussion

Par rapport à l'approche de planification utilisant les préimages, notre méthode a l'avantage de prendre en compte, en plus des erreurs sur la commande, des erreurs sur le modèle de l'environnement.

Dans la méthode utilisée par [Ala 94], le robot se rapproche d'un segment d'obstacle jusqu'à ce qu'il soit en contact. Il doit longer ce segment jusqu'à rencontrer une de ces extrémités. Dans notre approche, lorsque le robot se rapproche trop près d'un segment d'obstacle, il cherche à s'en éloigner pour éviter la collision. Le robot n'a pas besoin dans cette situation de longer ce segment.

Dans l'approche utilisant les cartes locales, le modèle de l'environnement est représentée par une grille. Dans chacune des grilles, le robot peut déceler des informations sur l'environnement. Les régions qui sont utilisées dans notre méthode regroupent toutes les cellules de la grille dans lesquelles le robot perçoit la même information sur l'environnement.

Conclusion générale

Conclusion

Nous avons présenté une nouvelle approche pour la planification de trajectoire d'un robot mobile à partir des informations provenant exclusivement de capteurs extéroceptifs, en particulier des capteurs ultrasons. On a évité l'utilisation de l'odométrie pour éviter l'accumulation des erreurs en position à chaque déplacement du robot.

Un modèle de l'environnement a été créé pour :

- obtenir a priori, l'information que le capteur est susceptible de produire lorsqu'il scrute une scène bien précise, comme un ou plusieurs segments d'obstacles, à partir d'une position donnée.
- reconnaître la région dans laquelle est produite l'information relative à un ou plusieurs segments d'obstacles.
- mettre en relation les régions adjacentes dans lesquelles le capteur tournant perçoit des informations distinctes sur les segments d'obstacles.
- trouver un chemin qui puisse mener le robot d'une configuration initiale vers une configuration finale à travers le plus petit nombre de régions représentant l'information sur les segments d'obstacle.

Le nombre de régions à représenter dans le modèle, selon la répartition des composantes de l'environnement, est relativement petit par rapport aux nombres de cellules utilisées dans d'autres méthodes de modélisation comme les quadtrees ou les NMV. Notre méthode utilise moins d'espaces mémoires pour le modèle.

Le temps nécessaire pour construire un graphe de noeuds permettant de réaliser la liaison entre les régions du modèle est inférieur à 1s.

Une commande a été appliquée au robot pour permettre de minimiser les écarts entre les mesures que l'on souhaite obtenir dans le modèle et celles obtenues par les capteurs en temps réel. Le robot est commandé par sa vitesse angulaire et linéaire. L'intérêt de la méthode utilisée pour commander le robot est de pouvoir connaître à tout moment l'orientation du robot par rapport à un référentiel local, c'est à dire de savoir dans quelle direction il s'oriente. Cette orientation est calculée à partir des informations qui sont perçues sur l'environnement. Les erreurs sur l'orientation du robot ne s'accumulent pas en fonction du temps, elles dépendent uniquement de la précision des mesures effectuées par les capteurs sur les segments d'obstacles. A n'importe quelle moment les erreurs sur l'orientation peuvent s'annuler. Des exemples ont illustrés le fait que la commande du robot est robuste par rapport aux erreurs sur le modèle de l'environnement.

L'application de la planification de trajectoire pour le projet V.A.H.M nous conduit à émettre des réflexions susceptibles d'apporter des améliorations ainsi que d'offrir des perspectives sur des nouvelles recherches.

Lorsque le robot doit passer d'une région T_1 vers une autre région T_2, il se trouve à un moment donné situé entre les deux régions. Dans cette situation, si des capteurs statiques sont installés sur le robot, une partie de ces capteurs va se trouver à l'intérieur de la région T_1, alors que l'autre partie sera présente dans la région T_2. Les capteurs qui sont utilisés dans la région T_1 perçoivent des informations relative à cette région, alors que les autres qui sont utilisés dans la région T_2 perçoivent des informations relative à cette dernière.

Dans le cas de la disposition des capteurs autour d'un robot réel, que les mesures qui sont obtenues par les capteurs dans une région du modèle sont différentes de celles effectuées dans une autre région adjacente. Parmi les deux régions T_1 et T_2, il y en a une dans laquelle les capteurs du robot détectent un segment d'obstacle, de plus que dans l'autre. Lorsque le robot se situe à un moment donné entre les deux régions, les mesures des capteurs qui se situent dans la première, ne peuvent pas être calculées par rapport au point d'origine P_c qui se situe dans l'autre région. Seule les mesures des capteurs effectuées dans la région dans laquelle se situe le point P_c sont prises en compte. Dans cette situation, on a un problème d'identification des capteurs qui se situent dans chacune de ces régions.

Les régions du modèle ont étés créées à la base pour représenter les mesures d'un capteur tournant. La présence d'un capteur statique du robot dans la région T_1 ou T_2 dépend de

l'orientation de celui-ci lorsqu'il se situe entre les deux régions. Elle dépend aussi de la taille du robot.

Une solution qui permet d'utiliser les capteurs statiques du robot dans le modèle consiste à changer la forme des régions du modèle en fonction de l'orientation du robot. L'orientation du robot ne peut pas être connue avant d'effectuer la planification lorsque celui-ci se situe entre deux régions. Cette orientation ne peut être calculée qu'au moment où le robot est situé entre ces deux régions. Les régions du modèle doivent changer en permanence de forme en fonction de l'orientation du robot. Les régions peuvent être représentées en trois dimensions, au lieu de deux dimensions comme on l'a fait, pour tenir compte de l'orientation du robot. Cette méthode se rapproche de l'espace des configurations utilisés en géométrie algorithmique.

Lorsque le robot est en mouvement à l'intérieur d'une région, son déplacement est commandé en vitesse angulaire et linéaire. Ces deux vitesses dépendent des variables $\dot{e}$ et $\ddot{e}$ calculées à partir de l'erreur e de mesures obtenues dans l'environnement du robot par rapport à celles du modèle. $\dot{e}$ et $\ddot{e}$ représentent respectivement la vitesse et l'accélération de e. Pour calculer le déplacement angulaire du robot par rapport à une configuration donnée, on utilise les valeurs de $\dot{e}$ et $\ddot{e}$ qui ont été calculées par rapport à la configuration précédente pour choisir le sens du déplacement angulaire du robot.

Par contre, si le robot se situe à la configuration initiale, $\dot{e}$ et $\ddot{e}$ ne peuvent pas être calculées par rapport à la configuration précédente puisque celle-ci n'existe pas. A ce moment le sens de déplacement du robot est choisi de manière aléatoire. Si le robot choisit de partir dans le mauvais sens, il s'éloigne du but. A la configuration suivante du robot, on aura $\dot{e}>0$, le robot doit donc faire demi-tour pour s'orienter dans la bonne direction.

Lorsque le robot passe dans une nouvelle région faisant partie de son chemin, il se trouve à la première configuration dans cette région. $\dot{e}$ et $\ddot{e}$ doivent à ce moment être calculées par rapport à la configuration précédente qui se trouve dans la région dans laquelle le robot se situe juste avant de passer dans la nouvelle. Les valeurs calculées de $\dot{e}$ et $\ddot{e}$ dépendent des mesures qui sont effectuées sur les segments d'obstacle. Comme ces mesures sont différentes d'une région à une autre, les valeurs calculées de $\dot{e}$ et $\ddot{e}$ dans une région ne peuvent pas être prise en compte pour déterminer le sens de déplacement angulaire du robot situé dans une autre. Le sens de déplacement angulaire du robot est choisi de manière aléatoire, à chaque fois

que le robot se situe à la première configuration atteinte dans une région faisant partie de son chemin.

Une solution qui permet de choisir le sens de déplacement angulaire du robot dans ces cas, est de se servir de la position de la configuration de référence (sous-but) située dans la région suivante sur le chemin du robot. Si on connaît la position de cette configuration, on peut obtenir l'orientation de celle-ci par rapport à celle du robot. On fait le calcul des orientations que pourrait avoir le robot par rapport à la configuration de référence, dans les deux sens de déplacement angulaire, et on choisit le sens pour lequel l'orientation du robot serait la plus proche de cette configuration.

Lorsque la planification de trajectoire du robot est effectuée en simulation, l'environnement dans lequel il se déplace est composé d'un ensemble de pixels. Chaque configuration du robot est située sur un pixel. Dans notre cas, le nombre total de pixels utilisés est de 256×256. Nous étions forcé d'utiliser cette résolution afin d'être compatible avec le module de lancé de rayon sur lequel nous nous sommes appuyés. Si le robot doit effectuer un déplacement linéaire de 1 pixel (ou 1 pas), il doit se déplacer vers un des 8 pixels adjacents à celui dans lequel il est situé. Il peut choisir uniquement entre 8 directions. Dans ce cas, le robot ne peut pas réaliser de manière précise le déplacement angulaire déterminé par la commande. Cela introduit des erreurs par rapport à l'orientation que doit avoir le robot en réalité. Comme dans cette situation, le robot est limité par le nombre de directions qu'il peut prendre, le mouvement du robot n'est pas fluide. La trajectoire du robot en simulation est moins lisse que dans la réalité.

Si le déplacement linéaire augmente, le robot peut s'orienter vers un nombre plus élevé de pixels. Il aura plus de possibilités pour choisir la direction qu'il doit prendre. On a intérêt à effectuer en simulation, des déplacements linéaires élevés (à partir de pas=3) .

L'état actuel du travail effectué est une étude de faisabilité de la méthode. Celle-ci a été développée en simulation. Le transfert vers le fauteuil va induire encore de nombreux problèmes à résoudre.

Annexe

Annexe A

Présentation du logiciel

L'ensemble des algorithmes présentés dans ce mémoire ont été implanté dans un logiciel qui permet de les utiliser de manière indépendante ou dans une tâche particulière. L'écriture des programmes a été réalisé en langage C++ sous l'environnement *Windows*.

Le développement des programmes sous *Windows* a permis une gestion de la mémoire beaucoup plus efficace que sous l'environnement Dos. De plus, le développement de programmes sous cet environnement nécessite une programmation très rigoureuse qui rend les programmes beaucoup plus compréhensibles et plus facilement réutilisable par autrui (notamment si on utilise une programmation orientée objet).

Le logiciel se décompose en 5 modules :

(1) Un module qui permet l'édition et la modélisation d'environnements géométriques :
- représentation des obstacles dans l'environnement du robot.
- représentation des régions constituées à partir des informations perçues dans l'environnement (segments d'obstacles) par le robot.

(2) Un module pour la construction d'un graphe permettant de mettre en correspondance les régions adjacentes crées à partir des informations sur les segments d'obstacles.

(3) Un module qui permet la planification de la trajectoire du robot, en appliquant l'algorithme exhaustif de Dijkstra.

(4) Un module qui permet de scruter un environnement réel au moyen des capteurs ultrasons pour obtenir les informations nécessaires pour le modèle et en temps réel pour pouvoir commander par la suite le robot.

(5) Un module qui permet la simulation de la commande du robot.

Annexe B

Fonctions de Windows utilisées pour créer et réaliser des opérations sur les régions.

La création de régions polygonales sous *Windows* comme des obstacles ou des zones représentant des informations sur l'environnement, est réalisée à partir de la fonction :

CreatePolygoneRgn(const POINT FAR*, INT)

où l'argument de type :
const POINT FAR* est l'adresse du tableau contenant les sommets de la région polygonale qui doit être crée.
INT est le nombre de sommets de la région.

La fonction retourne le *handle* (adresse) de la région crée si la fonction s'est bien déroulée et NULL dans le cas contraire.

Les opérations de combinaison entre deux régions, comme l'intersection ou l'union sont effectuées par l'intermédiaire de la fonction :

CombineRgn(hrgn1, hrgn2, hrgn3, TypeCombinaison)

où l'argument de type :

hrgn1 est le *handle* de la région qui est issue de la combinaison entre deux régions sources
hrgn2 est le *handle* de la première région source
hrgn3 est le *handle* de la deuxième région source
TypeCombinaison est le type de combinaison réalisée entre les deux régions sources

Les types de combinaisons sont :

- **RGN_AND** qui constitue l'opérateur d'intersection entre deux régions sources.
- **RGN_COPY** est un opérateur qui permet de réaliser une réplique de la région source représentée par le handle hrgn2. Dans ce cas le handle hrgn3 de la deuxième région source n'est pas utilisé.
- **RGN_DIFF** permet de créer une région représentée par le handle hrgn1 qui est constituée de la partie de la première région source (handle hrgn2) qui ne fait pas partie de la deuxième région source (handle hrgn3).
- **RGN_OR** permet d'obtenir l'union des deux régions sources.
- **RGN_XOR** permet d'obtenir l'union des deux régions sources mais sans la partie qui appartient à ces deux régions.

La fonction **CombineRgn()** est utilisée après que les trois régions représentées par les handles hrgn1, hrgn2 et hrgn3 ont été crées par l'intermédiaire de la fonction **CreatePolygoneRgn()**.

La fonction **CombineRgn()** retourne quatre valeurs entiers :

- ERROR qui est égale à 0. Dans ce cas, la fonction ne permet pas la combinaison entre deux régions sources.
- NULLREGION qui est égale à 1. Les deux régions sources n'ont aucune partie en commun et sont donc complètement séparées
- SIMPLEREGION qui est égale à 2. Les deux régions sources ont des régions simples en commun comme un point, une droite ou un rectangle.
- COMPLEXEREGION qui est égale à 3. Les deux régions sources ont des régions complexes en commun, c'est à dire des polygones.

La fonction **DeleteObject(hrgn)** permet d'effacer un objet, comme une région, de la mémoire en libérant tout stockage du système associé avec l'objet. Le handle de la région hrgn est utilisée à l'entrée de la fonction pour réaliser cette opération.

Références bibliographiques

Références Bibliographiques

[Ala 94] R. Alami and T. Siméon. *Planning robust motion strategies for a mobile robot*, IEEE International Conference on Robotics and Automation, vol. 2, pp. 1312-1318, San Diego, California, May 1994.

[Ahu 95] J. M. Ahuactzin, E. Mazer et P. Bessière. L'algorithme fil d'Ariane, Revue d'intelligence artificielle, Vol. 9, n°1, pp. 7-34, 1995.

[Ata 98] A. Atassi and A. Pruski. *Sensor based model for mobile robot path planning. CESA'98, pp. 285-290, Tunis,1998.*

[Bar 90] J. Barraquand and J.C. Latombe. *A Monte-Carlo Algorithm for Path Planning with many degrees of freedom. IEEE International Conference on Robotics and Automation*, 1990.

[Bar 95] J. Barraquand and P. Ferbach. *Motion Planning with Uncertainty : The Information Space Approach*, IEEE International Conference on Robotics and Automation, pp. 1341-1348, Nagoya, 1995.

[Bel 57] R. Bellman. *Dynamic programming*, Princeton University Press, Princeton, New Jersey, 1957.

[Ber 87] Bertsekas, *D. P. Dynamic Programming : Deterministic and Stochastic Models.* Prentice-Hall Inc., Englewood Cliffs, N. J., 1987.

[Bou 95] B. Bouilly, T. Siméon and R. Alami. *A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty*, IEEE International Conference on Robotics and Automation, pp. 1327-1332, Nagoya, 1995.

[Bour 93] G. Bourhis, K. Moumen, P. Pino, S. Rohmer and A. Pruski. *Assited navigation for a powered wheelchair*, Proceedings of the I.E.E.E System Man and Cybernetics Conference, Novembre 1993, Le Touquet, France.

[Bour 98] G. Bourhis, Y. Agostini. *Man-machine cooperation for the control of an intelligent powered wheelchair*, Journal of Intelligent and Robotic Systems, Special issue on « Mobile Robots in Health Care Services », vol 22, 1998, pp 269-287.

[Bre 65] J.E. Bresenham. *Algorithm for computer control of digital plotter*, 1965.

[Bro 82] R. A. Brooks. *Symbolic error analysis and robot planning*, International journal of robotics research, 1(4), 1982.

[Col 94] I. Collin, D. Meizel, N. Le Fort, and G. Govaert. *Local map design and task function planning for mobile robots. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, IROS'94, 1994

[Col 95] Ivan Collin, Planification de tâches-robots pour robots mobiles en environnement structuré, thèse soutenue à l'Université de Technologie de Compiègne, le 6 Janvier 1995, Compiègne.

[Cor 93] L. Cordewener, D. Meizel. Adaptation en ligne de la vitesse d'exécution de taches en robotique mobile. Revue française d'intelligence Artificielle, 7(4) : 465-480, 1993.

[Cou 98] A. Courcelle, O. Horn. *Ultrasonic data representation : application to mobile robots localisation*, Proceedings of the 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, Victoria, B.C., Canada October 1998.

[Dij 59] E. W. Dijkstra. *A note on Problems in connexion with graphs, Numerische Mathematik 1*, 269-271 (1959).

[Dun 82] O. Dunlaing, C .K. Yap. *A retraction method for planning the motion of a disc, journal of algorithms*, vol. 6, pp. 104-111, 1982

[Erd 86] M. Erdmann. *Using backprojections for fine motion planning with uncertainty*, International Journal of Robotics Research, 5(1), pp. 19-45, 1986.

[Fra 98] T. Fraichard and R. Mermond. *Path planning with uncertainty for car-like robots, IEEE International conference on robotics and automation*, pp. 27-32, Leuven, May 1998.

[Gil 66] *An iterative procedure for computing the minimum of a quadratic form on a convex set* , in SIAM J. Control, volume 4, n°3, pp. 61-80, 1966.

[Hab 94] O. Habert, O. Horn, and A. Pruski. *Distance Computing in the Multivalue Number Space*, Proceedings of the I.E.E.E International Conference on Systems, Man and Cybernetics, October 1994, San Antonio, Texas, Vol 3, pp. 2231-2236.

[Hab 95] O. Habert. Thèse de Doctorat sur la modélisation dynamique d'un environnement intérieur pour robot mobile, soutenue à Nancy le 29 mai 1995.

[Har 68] P. Hart, N. J. Nilsson and B. Raphael. *A formal basis for the heuristic determination of minimum cost paths, IEEE transaction on systems science and cybernetics*, vol. 4, pp 100-107, 1968.

[Jun 96] D. Jung, K. K. Gupta, *Octree-based hierarchical distance maps for collision detection*, in Proceeding of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, Minnesota, pp. 454-459, avril 1996.

[Kan 86] K. Kant and S.W. Zucker. *Toward efficient trajectory planning : path velocity decomposition*, International journal of robotics research, vol. 5,pp. 72-89, 1986

[Kana 91] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. *A stable tracking control method for a non holonomic mobile robot.* In IEEE/RSJ International Workshop on Intelligent Robots and Systems, pages 1236-1241, 1991.

[Kha 80] O. Khatib. Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles, thèse de l'école nationale supérieure de l'aéronautique et de l'espace, Toulouse, France, 1980.

[Kha 86].O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots., The International Journal of Robotics Research, 5(1), pp 90-98, 1986.

[Kho 88] P. Khosla, R. Volpe, Superquadratic Artificial Potentials for Obstacle Avoidance and Approach IEEE Conference on Robotics and Automation, pp 1778-1784 (1988).

[Lam 94] A. Lambert et N. Le Fort-Piat. Génération de cartes locales pour la planification de fonctions de tâches, rapport de D.E.A, Juillet 1994.

[Lam 97] A. Lambert and N. Le Fort-Piat. *Local map design for local navigation, International Conference on Advanced Robotics*, pp. 817-823, Monterey, July 1997.

[Lam 98] Alain Lambert. Planification de tâches sûres pour robot mobile par prise en compte des incertitudes et utilisation de cartes locales. Thèse soutenue à l'Université de Technologie de Compiègne, Sept 1998, Compiègne.

[Lat 91] J.C. Latombe. *Robot motion planning, Kluwer Academic Publishers*, 1991.

[Lau 79] J.L. Laurière . Eléments de programmation dynamique, Recherche opérationnelle appliquée, Gauthier-Villars, 1979.

[Leo 90] J. J. Leonard, H. F. Durrant-White, I. Cox. Dynamic Map Building for an autonomous Mobile Robot, The International Journal of Robotics Research, vol 11, n°4, August 1992.

[Leo 92] John J. Leonard, Hugh F. Durrant-White Directed Sonar Sensing for Mobile Robot Navigation , Kluwer Academic Publishers,1992.

[Loz 76] T. Lozano-Perez. *The design of a mechanical assembly system*, AI Memo 397, AI Laboratory, MIT, Cambridge, 1976.

[Loz 81] T. Lozano-Perez. *Automatic planning of manipulator transfer movements, IEEE Transaction on Systems, Man and Cybernetics*, 11(10), pp. 681-698, 1981.

[Miu 97] Jun Miura, Yoshiaki Shirai, *Vision and Motion Planning for a Mobile Robot under Uncertainty*, The International Journal of Robotics Research, vol. 16, n°6, December 1997, pp. 806-825, © 1997 Massachusetts Institute of Technology.

[Naj 94] J. Najéra, F. De la Rosa and C. Laugier. *Planning robot motion strategies under geometric uncertainty constraints*, International conference on intelligent robots systems (IROS), 1994.

[Nil 69] N. J. Nilsson. *A mobile automation : an application of artificial intelligence techniques, Proceedings of the First International Conference on Artificial Intelligence*, Washington D.C., pp. 509-520, 1969.

[Nsi 98a] Nsingi Masunga, P. Arnould. *Autonomous mobile robot navigation using the odometer and the ultrasonic sensors*, IEEE Roman'98 International Workshop on robot and human communication, Japon, Octobre 1998

[Nsi 98b] Nsingi Masunga, P. Arnould. *Mobile robot navigation by multi-sensor integration*, Asia-Europe Congress on mechatronics, Japon, Octobre 1998.

[Pag 94] Lance A. Page and Arthur C. Sanderson. *Sensing and control uncertainty fields in sensor based motion planning, in Proc. Sensor Fusion VII*, pages 226-236, 1994. SPIE vol. #2355.

[Pag 95] Lance A. Page and Arthur C. Sanderson. *Robot motion planning for sensor-based control with uncertainties, IEEE International conference on robotics and automation*, pp 1333-1340, Nagoya, 1995.

[Pay 97] P. Payeur, P. Herbert, D. Laurendeau, C. M. Gosselin, *Probabilistic octree modeling of a 3D dynamic environment*, IEEE International conference on robotics and automation, Albuquerque, New Mexico, avril 1997.

[Pig 94] P. Pignon. Structuration de l'espace pour une planification hiérarchisée des trajectoires de robots mobiles. Ph.D. Dissertation, Laboratoire d'Automatique et d'Analyse des Systèmes (LAAS), Toulouse, 1994.

[Pru 91] A. Pruski. *Multivalue Coding for image and solid processing*, Revue International de CFAO et d'Infographie, Vol 6, n°2, pp 135-152 (1991).

[Pru 92] A. Pruski. *Multivalue Coding : Application to autonomous robots*, Robotica, Vol 10, 125-133, (1992).

[Pru 97] A. Pruski, S. Rohmer. *Robust Path Planning for Non-holonomic Robots*, Journal of Intelligent and Robotic Systems 18 : 329-350, 1997 Kluwer Academic Publishers.

[Roh 93] S. Rohmer. Modélisation d'Environnement par Nombres Multivaleurs, Thèse de l'université de NANCY I, Novembre 1993, France.

[Sam 80] H. Samet, *Region Representation, Quadtrees from Binary Arrays, Computer Graphics and Image Processing*, vol 13, pp 88-93, 1980.

[Sam 91] C. Samson, M. Le Borgne, and B. Espiau. Robot Control *: The Task Function Approach*. Oxford University Press, Oxford, 1991.

[Sam 92] C. Samson. *Path following and time varying feedback stabilisation of a wheeled mobile robot*, Proceedings of the ICARCV'92, Singapore, pp. RO-13.1.1-RO-13.1.5, 1992.

[Tak 92] H. Takeda et J.C. Latombe, *Sensory Uncertainty Field for Mobile Robot Navigation, IEEE Conference on Robotics and Automation*, Nice, France, 1992, pp2465-2472.

[Tak 94] H. Takeda, C. Facchinetti, J.C. Latombe. *Planning the motion of a Mobile Robot in a Sensory Uncertainty Field, IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 16, n°10, Oct. 94.

[Tay 76] R. H. Taylor. *Synthesis of manipulator control programs from task level specifications*, AI Memo 228, AI Laboratory, Stanford, 1976.

[Vla 98] N.A. Vlassis and P. Tsanakas. *A Sensory Uncertainty Field Model for Unknown and Non-stationary Mobile Robot Environments*, Leuvin, May 1998.

[Whi 77] D. E. Whitney, *Force feedback control of manipulator fine motion*, J. Dyn. Syst. Measurement Contr. 98, pp. 91-97.

Résumé

Dans notre mémoire de thèse, nous présentons une nouvelle approche de planification de trajectoire robuste pour un robot mobile, basée entièrement sur les informations provenant des capteurs d'environnement. Les travaux sont effectués dans le cadre du projet V.A.H.M. (véhicules autonomes pour handicapés moteurs). Nous avons cherché à limiter l'utilisation de l'odométrie pour laquelle le calcul de la position du robot introduit une accumulation d'erreurs lors de son déplacement. Des régions sont présentées dans le modèle de l'environnement, dans lesquelles le robot perçoit des informations distinctes sur les segments d'obstacles. Un graphe de noeuds est construit, afin de mettre en relation les régions adjacentes dans lesquelles le robot perçoit un changement d'information. A partir de ce graphe, on utilise un algorithme de planification qui choisit le chemin le plus robuste, en terme de minimisation du nombre de régions traversées, entre une configuration initiale et finale du robot. La dernière étape consiste à proposer une loi de commande en vitesse angulaire et linéaire qui permet de mener le robot d'une région vers une autre. Le système est robuste par rapport aux erreurs du modèle de l'environnement et de la commande.

Abstract : - Our thesis introduces a new approach to robust path planning for mobile robots. This approach is entirely based on ultrasonic sensors information, and avoids the use of odometry, which leads to the accumulation of errors resulting from the calculation of the robot position. We have proceeded as follows: For each segment of obstacle detected by the mobile robot sensors, we create a free space region. A node graph is used to represent the regions with their links. With this graph as a basis, we use a planning algorithm which chooses the required path. The final stage consists in finding for the robot motion a robust control, as regards the environment model errors (uncertainties). This approach could contribute in practise to a control system for indoor robot motion, which offers increased accuracy to an economical ultrasound device.