Université de Nancy I — UFR STMIA

Centre de Recherche en Informatique de Nancy

MÉTHODES GÉOMÉTRIQUES ET ALGORITHMIQUES POUR LA PLANIFICATION DE TRAJECTOIRES

THESE

pour l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE NANCY I
Spécialité informatique

Henri AMET

Soutenue le 21/12/1990

Devant la commisssion d'examen :

Président	Roger Mohr
Rapporteurs	Jean-Paul Haton
	Kurt Mehlhorn
Examinateurs	Jean-Daniel Boissonnat
	Pierre Lescanne
	René Schott

Université de Nancy I

UFR STMIA

Centre de Recherche en Informatique de Nancy

MÉTHODES GÉOMÉTRIQUES ET ALGORITHMIQUES POUR LA PLANIFICATION DE TRAJECTOIRES

THESE

pour l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE NANCY I
Spécialité informatique

Henri AMET

Soutenue le 21/12/1990

Devant la commisssion d'examen :

Président	Roger Mohr
Rapporteurs	Jean-Paul Haton
	Kurt Mehlhorn
Examinateurs	Jean-Daniel Boissonnat
	Pierre Lescanne
	René Schott

à Nelly, Marie-France et Valérie,

à mes parents,

à ma petite soeur,

à ma grand-mère,

Remerciements

Je tiens à remercier René Schott, mon directeur de thèse, qui m'a communiqué tout au long de ces 3 années, sa passion de la recherche. Particulièrement au courant de l'activité scientifique du domaine, il a mis constamment à ma disposition les plus récentes publications. Il m'a permis d'avoir des contacts fructueux avec des laboratoires extérieurs. Il m'a encouragé de façon permanente et ne s'est pas opposé à ma démarche personnelle. Enfin, sa gentillesse et sa patience m'ont rendues ces 3 années agréables.

Je remercie Jean-Daniel Boissonnat pour m'avoir acceuilli à plusieurs reprises dans son équipe. L'unité de recherche de Sophia Antipolis reste pour moi un exemple. Je le remercie pour le travail réalisé en commun avec René Schott et moi même concernant le diagramme de Voronoï dynamique d'un ensemble de segments. Je le remercie enfin de faire partie de mon jury, lui qui a un emploi du temps si chargé.

Je remercie Francis Avnaim pour les discussions fructueuses que j'ai pu avoir avec lui. Ses travaux me sont toujours apparus comme exemplaires de part leur qualité théorique et pratique. Son travail concernant le déplacement exact d'un polygone m'a d'ailleurs inspiré. Sans lui, le chapitre 2 n'aurait certainement pas vu le jour ...

Je remercie Kurt Mehlhorn pour m'avoir lui aussi accueilli dans son laboratoire et pour accepter d'être rapporteur de cette thèse, sachant qu'il est particulièrement occupé.

Je suis très honoré que Jean-Paul Haton, spécialiste incontesté de l'intelligence artificielle, ait accepté d'être rapporteur de ma thèse.

Je remercie chaleureusement Pierre Lescanne de faire partie de mon jury. Il s'est toujours montré très intérressé par mon travail.

Je remercie l'équipe Eureca. J'y ai toujours trouvé des conditions de travail parfaites. Je n'oublierai pas non plus l'INRIA dont le support financier m'a permis de mener à bout ce travail.

Enfin, je réserve pour la fin, des remerciements particuliers à 3 personnes qui m'ont marquées dans leur personnalité. Il s'agit tout d'abord de Roger Mohr, qui fait partie de mon jury, et à qui je voue une admiration sincère pour ses qualités morales et intellectuelles.
Il s'agit ensuite d'Alain Filbois avec qui j'ai eu des discussions fructueuses et passionnées. Enfin, j'ai une pensée particuliére pour Saad Benachi, mon collègue de bureau, qui s'est toujours montré patient, droit et sincère.

Résumé

Cette thèse s'intéresse à la recherche d'une trajectoire sans collision pour un système robotique au sein d'un environnement connu à priori ou inconnu. Ce travail s'inscrit dans le cadre général de l'algorithmique géométrique, nouvelle branche de l'informatique qui traite des problèmes de nature géométrique.

Nous présentons au chapitre 1 un rapide aperçu des différents axes de recherche dans le domaine de la planification de trajectoires.

Nous présentons ensuite au chapitre 2 un algorithme exact résolvant le déplacement en translation et rotation d'un polygone quelconque au sein d'un environnement composé de polygones quelconques.

Le chapitre 3 traite d'un outil très utilisé en planification de trajectoires : le diagramme de Voronoï. Nous présentons rapidement les algorithmes existants pour le calcul du diagramme sur des points. Nous nous intéressons ensuite au calcul dynamique du diagramme d'un ensemble de segments. Puis nous présentons des algorithmes concernant le calcul d'un diagramme discret pour différentes configurations : diagramme d'objets quelconques dans le plan et diagramme généralisés au cas d'un polygone au sein d'un ensemble de polygones.

Nous proposons au chapitre 4 une application pratique de l'utilisation des diagrammes de Voronoï : ils fournissent une trajectoire guide pour des déplacements en translation et rotations.

Le chapitre 5 propose une solution algorithmique au même problème que celui traité au chapitre 2 mais dans un univers discret et à l'aide d'opérations spécifiques à cet univers. Nous montrons qu'elles possèdent de nombreux avantages : des qualités dynamiques et locales ainsi qu'une grande sécurité d'emploi.

Les solutions proposées sont originales. L'effort a essentiellement porté sur le developpement de logiciels. Ce travail a fait l'objet de plusieurs publications (voir la bibliographie).

Mots clés :

Géométrie algorithmique, planification de trajectoires, robots mobiles, diagrammes de Voronoï.

Table des matières

Chapitre 1

Définition générale de la planification de trajectoires et grandes directions de recherche

1.1 Besoins actuels en robotique

Les futures générations de robots devront être capables de réaliser des tâches beaucoup plus complexes que celles qui leurs sont demandées aujourd'hui. La plupart des systèmes actuels sont dépendants de l'homme. Une tâche n'est réalisée que si elle est complétement décortiquée par lui. On voudrait pouvoir exprimer un travail à faire en termes très généraux. Par exemple, spécifier uniquement l'assemblage final d'un certain processus. Le robot serait alors capable de déterminer les étapes intermédiaires. Ces nouvelles capacités peuvent être rangées dans 3 catégories :

• Perception : le robot doit être capable de percevoir son environnement à l'aide d'une grande variétés d'organes (organes tactiles, visuels, de proximité).

• Planification : le robot doit concevoir la suite de sous-tâches menant à la tâche finale.

• Commande : chaque sous-tâche doit effectivement pouvoir être réalisée par le robot. Par exemple, saisir un plateau à l'aide d'une pince.

Dans cette étude, nous ne parlerons que de planification et celle-ci ne concernera que la production de trajectoires sans collision d'un systéme robotique au sein d'un certain environnement. Les outils nécessaires à l'étude de la planification sont variés : ils regroupent la géométrie classique, la topologie, la géométrie algébrique, l'analyse combinatoire et dans le domaine informatique, la géométrie algorithmique, l'analyse numérique ... Les 2 grands axes de recherche concernent d'une part, l'approche mathématique exacte qui permet au moins de prendre conscience des difficultés du problème, et d'autre part, une approche avec heuristiques, qui fournit dans bien des cas, des solutions viables. Ces 2 grands axes sont abordés dans cette étude.

1.2 Définition générale de la planification de trajectoires et notion d'espace libre

Soit I un système robotique composé d'un ensemble de parties rigides éventuellement reliées entre elles par des articulations. Certaines parties peuvent se mouvoir indépendamment des autres. Soit k le nombre de degrés de liberté du système. Un placement du système robotique peut donc être spécifié de façon non ambigüe par k réels. Cet ensemble de k-uplets est appelé *espace des configurations*. Les k réels représentent soit les positions de points particuliers de I soit des angles entre les différentes parties. Supposons que I soit libre de se mouvoir dans un espace à 2 ou 3 dimensions parmi un ensemble d'obstacles dont la géométrie est connue ou inconnue du robot. Appelons E cet ensemble d'obstacles. Les valeurs classiques de k vont de 2 (pour un robot composé d'une seule partie et qui se déplace en translation pure dans un espace à 2 dimensions) à 6 (pour un bras manipulateur). Ces valeurs peuvent éventuellement être plus grandes lorsqu'il s'agit de coordonner par exemple les déplacements d'un ensemble de plusieurs systèmes robotiques.
Le déplacement de I peut s'exprimer de la façon suivante :

Soit Z_1 le placement initial de I, soit Z_2 le placement final, déterminer s'il existe un déplacement continu entre Z_1 et Z_2 et si oui comment le réaliser.

Il existe un grand nombre de variantes à ce problème. Il se peut que l'environnement ne soit pas totalement connu du robot, que les obstacles ne soit pas fixes ou bien que l'environnement ne soit pas connu avec précision. Dans ce dernier cas, le robot devra prendre en compte ces imprécisions. Il se peut aussi que l'on désire une trajectoire optimale qui minimise par exemple le temps de réalisation ou la somme des déplacements. Précisons maintenant la notion d'*espace libre*.

Définition Soit k le nombre de degrés de liberté du système robotique I. L'espace libre EL est l'ensemble des k-uplets $(x_1,x_2,...,x_k)$ tels que les placements associés à chacun d'eux soient sans collision (ou libres) avec l'environnement.

Dans le cas du déplacement d'un robot rigide composé d'une seule partie et se déplaçant dans le plan, EL vaut l'ensemble des (x,y,α) tels que les placements associés soient libres. x et y fixent la position d'un point particulier de I et α fixe une certaine rotation de I par rapport à une rotation origine. El est donc ici un sous-ensemble de $\mathbb{R}^3$. Avnaim et Boissonnat donnent dans [A & B 88] une vue en perspective d'un tel sous-ensemble. Dans un premier temps, on peut observer que son allure n'est pas triviale.

On comprend mieux maintenant la nature de ce qu'est un déplacement sans collision. Il s'agit de rechercher dans l'espace libre une suite de déplacement élémentaires. Nous pouvons dès maintenant donner un résultat fondamental :

Résultat Il existe un déplacement continu de I entre les positions Z_1et Z_2 si et seulement si Z_1et Z_2 appartiennent à la même composante connexe de l'espace libre.

Une conséquence intéressante de ce résultat est qu'il suffit de calculer la composante connexe contenant la position de départ Z_1 et de déterminer ensuite si la position d'arrivée Z_2 y appartient. Il n'a pas encore été démontré que cette façon de faire réduisait la complexité générale du problème. Il s'agit la d'un problème ouvert fameux.

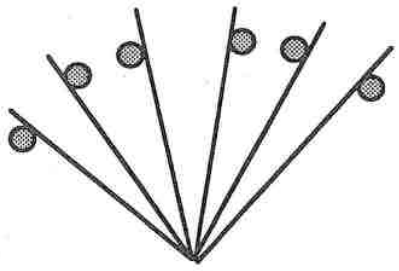
1.3 Complexité du calcul des trajectoires

Dans ce qui suit, nous supposerons que l'environnement est statique et connu du robot. L'espace libre EL est déterminé par un ensemble d'équations ou d'inéquations qui expriment le fait que tout placement appartenant à EL est libre. Soit *n* ce nombre d'équations ou d'inéquations. Celles-ci définissent des surfaces dans $\mathbb{R}^k$. Le bord de EL est constitué de facettes. Ces facettes s'intersectent et donnent naissance à des sommets. Ceux-ci, d'un point de vue concret, représentent des k-contacts entre le robot et son environnement. Un k-contacts est un positionnement de I pour lequel il y a k contacts distincts avec l'environnement. Le nombre de k-contacts donnent la complexité de EL. Essayons de dénombrer ce nombre de sommets. Chacun d'eux peut être l'intersection de k de ces n surfaces. Il peut donc y en avoir $C_{k,n}$ (nombre de combinaisons de k éléments parmi n) et on sait que $C_{k,n}$ se comporte comme n^k. On montre ainsi que la complexité topologique de EL est polynomiale en le nombre n de contraintes mais exponentielle en le nombre k de degrés de liberté. Une autre façon de se rendre compte de la complexité de EL est de considérer le système robotique suivant : I est constitué par k segments de longueur égale tous reliés en un point fixe O (I forme une sorte d'éventail). I a donc k degrés de liberté. L'environnement est constitué de n petits disques répartis en arc de cercle autour de O (voir figure 1). On remarque immédiatement que le nombre de k-contacts vaut n^k.

Le résultat fondamental est dû à Schwarz et Sharir (voir [S & S 82]). Il montre aussi par ailleurs que la complexité de la recherche d'une trajectoire est polynomiale en n mais doublement exponentielle en k.

Conclusion Ce résultat négatif conditionne les recherches en planification de trajectoires. 3 grands axes de recherche s'en déduisent. Le premier vise à concevoir des algorithmes pour résoudre le cas général (k quelconque). L'intérêt est surtout théorique car on sait qu'ils sont

irréalistes. Le second axe de recherche vise à concevoir des algorithmes exacts pour les cas où k est petit (k=2 ou 3). Ici, on cherchera à exploiter le plus possible les particularités de la situation. Le déplacement en translation et rotation d'un polygone dans le plan en est un exemple. Le troisième axe s'intéresse au cas où k est plus grand mais des heuristiques sont utilisées. Le cas typique est celui d'un bras manipulateur à 6 degrés de liberté.



C

Figure 1 : Complexité de l'espace libre

Dans cette étude, nous avons privilégié l'efficacité. Nous proposons des algorithmes appartenant aux deux dernières classes citées.

1.4 Les différentes approches

1.4.1 Les méthodes globales

Nous allons exposer ici les principales méthodes globales. On entend par méthode globale une méthode qui calcule un modèle de tout l'espace libre pour y rechercher ensuite une trajectoire à l'opposé des méthodes locales qui n'ont qu'une vue partielle de l'environnement.

• La méthode de Schwartz et Sharir

Elle est basée sur la décomposition cylindrique de Collins des ensembles semi-algébriques et sur l'algorithme de suppression des quantificateurs de Tarski. La première étape consiste à obtenir une formule générale algébrique à partir du système de contraintes. Cette formule est composée de sous-formules contenant des expressions polynomiales sur les k variables. La technique consiste à décomposer l'espace des configurations en un nombre fini de cellules connexes telles que pour chacune d'elles, les polynomes apparaissant dans la formule gardent un signe constant. Le nombre de cellules obtenues est doublement exponentiel en k. Ces cellules sont soit complétement dans l'espace des configurations interdites (complémentaire de l'espace libre dans l'espace des configurations) soit complétement dans l'espace libre. Il suffit ensuite de bâtir un graphe de connexité sur les cellules appartenant à EL. La recherche d'une trajectoire se fera en déterminant à quelles cellules appartiennent les configurations de départ et d'arrivée puis à rechercher dans le

graphe discret un plus court chemin. Cette méthode très générale est peu efficace du fait du très grand nombre de cellules obtenues (voir [S & S 82]).

• **La méthode de Canny**

Elle diffère complètement de la précédente. Le but est de construire un squelette S de l'espace libre (on dira aussi une rétraction) possèdant 2 propriétés essentielles :
-La propriété *d'atteignabilité* : elle dit que de tout placement Z de l'espace libre, il existe un déplacement continu et libre de Z vers un point de S.
-La propriété de *connexité* : elle dit que toutes les intersections entre S et les composantes connexes de EL sont non vide et connexes.
La recherche d'une trajectoire se fait ensuite en recherchant un chemin sur le squelette entre les positions W1 et W2 où W1 est un point de S atteint depuis Z1 et W2 un point de S atteint depuis Z2.
La complexité annoncée est en $O(n^k.\log n)$ ce qui améliore la complexité de l'algorithme de Schwartz et Sharir (voir [CAN 87] et [C & R 87]).
Nous allons donner maintenant un aperçu rapide des méthodes globales pour des cas particuliers.

• **Les méthodes directes**

On dit qu'une méthode est directe lorsqu'on peut calculer directement et simplement l'espace libre. Prenons l'exemple du déplacement en translation pure d'un segment S parmi un ensemble de polygones. Considérons un point particulier du segment. Appelons le s. L'espace des configurations interdites vaut la différence de Minkowsky entre les polygones constituant l'environnement E et le segment S. Cette différence est notée $E \Theta S$. Elle vaut l'ensemble des x-y (différence vectorielle) avec x appartenant à E et y appartenant à S lorsque celui-ci est ramené à l'origine (s et l'origine coïncident). On dit que $E \Theta S$ représente l'agrandi des obstacles par S (ou le dilaté). La différence de Minkowski est dans ce cas un ensemble de polygones. La recherche d'une trajectoire se fait donc entre les positions Z_1 et Z_2 au sein des polygones de Minkowski sur le graphe des géodésiques construit sur l'ensemble des sommets des polygones dans lequel les points Z_1 et Z_2 ont été ajoutés. Rappelons qu'une géodésique est une trajectoire qui minimise la distance parcourue entre ses 2 extrémités. Le graphe des géodésiques est un sous-graphe du graphe de visibilité. Les arcs ne pouvant à priori pas faire partie d'une plus courte trajectoire ont été retirés. Cette notion est très intuitive. Les arcs du graphe des géodésiques sont soit des chaînes convexes à la frontière des polygones soit des tangentes d'appui ou des tangentes séparatrices. La figure 2 montre les polygones de Minkowsky de 2 polygones ainsi que le graphe des géodésiques.

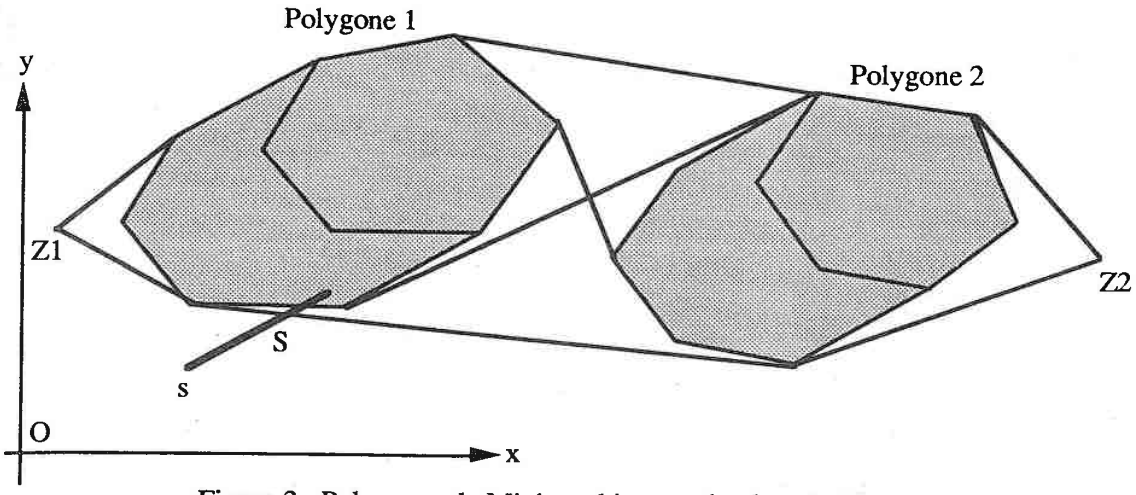


Figure 2 : Polygones de Minkowski et graphe des géodésiques

• Les méthodes par projection

Les idées exposées ici sont similaires à celles de la méthode de Schwartz et Sharir. Le but est dans un premier temps de bâtir un modèle de l'espace libre. Il suffit de fixer un degré de liberté parmi les k et de résoudre récursivement le sous-problème pour les k-1 autres degrés non fixés. Appelons y la variable fixée. On obtient ainsi en fonction de y une tranche de l'espace libre recherché. Cette tranche conserve une forme analytique constante en y, excepté pour un certain ensemble de valeurs dites critiques de y. Ces valeurs critiques sont calculées. Elles partitionnent l'espace libre en cellules connexes entre lesquelles il faut établir des relations d'adjacence. Un graphe de connexité est finalement obtenu. La recherche d'un chemin fournit la solution au problème de planification.

• Les méthodes par rétraction

Les idées exposées ici sont similaires à celles de la méthode de Canny. Le but est de rétracter l'espace libre en un sous-espace N de dimensions inférieure (habituellement 1) qui vérifie les propriétés d'atteignabilité et de connexité. Autrement dit, les positions Z1 et Z2 appartiennent à la même composante connexe de l'espace libre si et seulement si leurs rétractions sur N appartiennent à la même composante connexe de N. Si N est de dimension 1 alors le problème se réduit en une recherche dans un graphe.

Un bel exemple d'illustration de cette méthode est celui concernant le déplacement d'un disque parmi un ensemble de polygones. Ici, la rétraction est le diagramme de Voronoï de l'ensemble des polygones. Nous reviendrons longuement par la suite sur ce diagramme. Disons simplement que le diagramme est l'union des bords des cellules de Voronoï associées à chacun des polygones et que la cellule de Voronoï associée au polygone i est l'ensemble des points situés plus près du polygone i que de tout autre polygone. Lors des déplacements, le centre du disque se déplacera sur le diagramme. Il conviendra enfin de retirer au diagramme de Voronoï les parties pour lesquelles la

distance au plus proche polygone est inférieure au rayon du disque. La figure 3 donne un tel exemple de rétraction. Des arcs ont été rajoutés dans les parties concaves du polygone englobant.

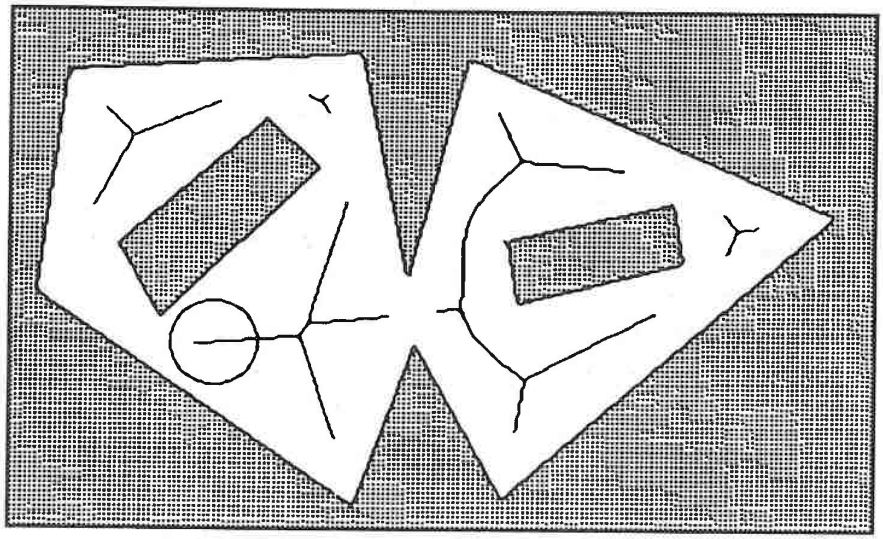


Figure 3 : Rétraction de l'espace libre par le diagramme de Voronoï
dans le cas du déplacement d'un disque

Conclusion Nous arrêterons là les descriptions des méthodes globales. Elles présentent l'inconvénient de calculer tout l'espace libre de l'environnement alors que le déplacement demandé peut n'affecter qu'une petite partie de l'environnement. A l'opposé, elles garantissent une trajectoire si celle ci existe. Nous allons aborder maintenant 2 méthodes locales : celle dite des potentiels et celle des contraintes.

1.4.2 Les méthodes locales

Les méthodes locales n'ont qu'une vue locale de l'environnement. Elles ont des avantages et des inconvénients complémentaires de ceux des méthodes globales. En particulier, elles ne demandent pas de mémoire alors que le stokage de l'espace libre pour les méthodes globales était particulièrement coûteux. Ces méthodes présentent aussi l'avantage de pouvoir prendre en compte un environnement évolutif. L'inconvénient majeur est bien sur la non garantie d'arrivée à l'objectif et plus précisemment le blocage possible dans des configurations concaves de l'environnement.

◦ La méthode des potentiels

La méthode des potentiels a été introduite par Khatib (voir [KHA 86] et [K & M 78]). Le sytème robotique I est plongé dans un champ de potentiel fictif U(q) valant la somme d'un potentiel attractif l'attirant vers l'objectif et de potentiels de répulsion émis par les obstacles. On a :

$U(q) = PA(q) + \Sigma_{i=1,m} U_i(q)$

où : PA(q) est le potentiel attractif centré vers l'objectif
$U_i(q)$ est le potentiel de répulsion engendré par le ième obstacle.

Les potentiels répulsifs ont une distance d'influence limitée, ce qui assure qu'un obstacle éloigné est sans effet sur le robot. On obtient la force fictive correspondant à U(q) en prenant les gradients des potentiels élémentaires. Les déplacements élémentaires sont obtenus en minimisant la force fictive.
La méthode des potentiels est particulièrement simple d'un point de vue algorithmique. Dans la réalité, il est cependant difficile d'ajuster les coefficients utilisés pour pondérer la force attractive et les forces répulsives. Son inconvénient majeur est le blocage du robot dans un minimum local.

• La méthode des contraintes

Le but est de traduire les interactions entre les solides suceptibles d'entrer en collision pour la configuration courante du système robotique en des contraintes sur la variation des paramètres de configuration. Une contrainte signifie que la distance entre un élément du robot et un obstacle ne doit pas décroître trop vite lorsqu'elle est inférieure à une certaine distance.
A chaque pas de discrétisation, les étapes suivantes seront effectuées :

- à partir du modèle du robot et de celui de l'environnement, calculer une vue courante des interactions entre solides.
- construire un modèle local de l'espace libre dans l'espace des configurations, qui traduit le contrôle des points de distance minimale entre les corps mobiles et les obstacles.
- se rapprocher de l'objectif en restant dans l'espace libre.

L'interaction entre un élément du robot et un obstacle se traduit par une contrainte affine dans l'espace des configurations. L'intersection des demi-espaces libres définis par ces contraintes constitue la vue locale de l'espace libre à l'instant considéré. Par ailleurs, à chaque pas de la discrétisation, le système robotique choisit le meilleur déplacement élémentaire en minimisant un certain ensemble de mesures. Voir [TOU 88] pour plus de détails.

1.4.3 Les méthodes mixtes

Les méthodes mixtes tentent de marier les avantages des méthodes globales et locales. Le but est de séparer le problème général de planification de trajectoires en un générateur local de trajectoires et un planificateur global basé sur un graphe de régions relativement grandes de l'espace des configurations. Etant donné les configurations initiales et finales, un algorithme classique de recherche d'un plus court chemin dans le graphe génère une liste de cellules adjacentes contenant ces 2 configurations. L'étape de base consiste, pour le robot, à pénétrer dans la cellule suivante de celle qu'il occupe à un instant donné. Cette cellule suivante est fournie par le planificateur global.

Des probabilités sont affectées à chaque transition élémentaire suivant la facilité de réalisation de celle-ci. En particulier, en cas de disposition concave des obstacles empêchant la transition, une probabilité plus faible lui est alors attribuée et le planificateur global propose une nouvelle suite de cellules dans le graphe modifié.
Le principal avantage de cette méthode est de ne nécessiter qu'une discrétisation grossière de l'espace des configurations. Cette approche conserve aussi la possibilité de prise en compte dynamique de l'environnement. Voir [F & T 87] pour plus de détails.

1.5 Les approches choisies dans ce travail

Le résultat de Schwartz et Sharir ne nous a pas incliné à concevoir des algorithmes généraux de planification de trajectoires. Il fallait soit concevoir des algorithmes exacts avec un petit nombre de degrés de liberté soit utiliser des heuristiques. Nous nous sommes restreint dans cette étude à la planification de trajectoires pour un mobile évoluant en translation et rotation dans le plan. Le nombre de degrés deliberté est donc 3. Nous nous sommes intéressés dans un premier temps à un algorithme exact puis nous nous sommes orientés vers des algorithmes utilisant des heuristiques afin d'améliorer les performances tout en conservant une bonne probabilité de succès. L'idée principale, dans ce cas, a été l'utilisation du diagramme de Voronoï des obstacles. Le calcul de celui-ci a fait l'objet d'efforts importants. Nous nous sommes intéressés ensuite à des algorithmes dans un espace discret. Ceux-ci nous ont semblé présenter des avantages du point de vue simplicité et robustesse. Certaines opérations deviennent particulièrement simples dans un espace discret.

1.6 Etude de certains aspects liés à l'algorithmique géométrique : l'aspect dynamique et le parallélisme

La conception d'algorithmes dynamiques est aujourd'hui une nécessité. Elle fait l'objet d'efforts important parmi les chercheurs. L'environnement d'un système robotique évolue continuellement et il ne saurait être question, à chaque modification, de reconstruire un modèle de l'espace libre. Nous avons vu que les méthodes locales présentent cet avantage mais nous aimerions adjoindre cette qualité aux méthodes globales. Nous proposons une construction dynamique du diagramme de Voronoï d'un ensemble de segments.
La parallèlisation des algorithmes est aussi un des axes importants de recherche en planification de trajectoires. Des machines multi-processeurs commencant à apparaître sur le marché, il est utile de concevoir de tels algorithmes. Nous avons réalisé l'implantation d'un algorithme de construction du diagramme de Voronoï sur un T_Node.

1.7 Effort particulier

La production de logiciels a fait l'objet d'un effort particulier tout au long de cette thèse. La plupart du temps a été occupée par cette activité. La validation d'une idée par un logiciel nous a semblé importante. En particulier, les algorithmes "baroques" (algorithmes et structures de données compliqués à souhait) ne résistent pas longtemps à l'épreuve de vérité qu'est la programmation. De plus, l'effort a porté sur la robustesse des algorithmes et des implantations. Produire du logiciel fragile n'est d'aucune utilité. L'ensemble des modules écrits compose le logiciel SPAT (Système de planification automatique de trajectoires). Chaque chapitre sera terminé par une présentation rapide du logiciel créé ainsi que par des copies d'écran des exécutions.

Chapitre 2

Déplacement en translation et rotation d'un polygone quelconque parmi un ensemble de polygones

2.1 Introduction

Etant donné 2 ensembles de polygones I et E avec m, respectivement n arcs, nous présentons un algorithme exact calculant le déplacement de I au sein de E étant donné une position initiale et une position finale de I. Le déplacement est possible en translation et en rotation. Ce travail représente une alternative aux travaux de F.Avnaim et J.D.Boissonnat sur le même sujet (voir [A & B 88]). Il représente une approche plus géométrique et intuitive du déplacement d'un polygone. L'algorithme final construit explicitement le bord de l'espace libre. Déplacer I au sein de E revient alors à rechercher un chemin dans un graphe. La complexité de l'algorithme final est en $m^3.n^3.\log(m.n)$ dans le pire cas. Cette méthode s'inscrit dans le cadre des méthodes de projection. Le paramètre figé sera ici l'angle fixant la rotation du polygone I.
Cet algorithme devrait se généraliser au cas du déplacement d'un splinegone au sein d'un ensemble de polygones. (Un splinegone étant un polygone dont les arêtes sont remplacées par des courbes telles que la surface entre une courbe et son arête respective est convexe). Enfin, cet algorithme a été implanté en langage C.

2.2 Définitions

Nous reprenons ici les notations et définitions de l'article de F.Avnaim et J.D.Boissonnat [A & B 88] ainsi que les notions de contacts simples, doubles et triples.

- Soient I1,...,Im les sommets de I.
- Soient i1=[I1,I2],...,im=[Im,I1] les arcs de I.
- Soient E1,...,En les sommets de E.
- Soient e1=[E1,E2],...,em=[Em,E1] les arcs de E.

Dans la suite, les sommets de I ou de E seront considérés dans l'ordre contraire des aiguilles d'une montre.

- On appelle *contact* un couple composé soit d'un sommet de I et d'un arc de E, soit d'un arc de I et d'un sommet de E. Le premier type sera qualifié de "vertex-edge" et le second de "edge-vertex".

Par la suite on dira que I et E possèdent un contact s'il existe un tel couple tel que le sommet du contact appartienne à l'arc du contact. On dira que I et E possèdent un double-contact s'il existe 2 tels couples, et qu'ils possèdent un triple-contact s'il existe 3 tels couples (voir la figure 1)

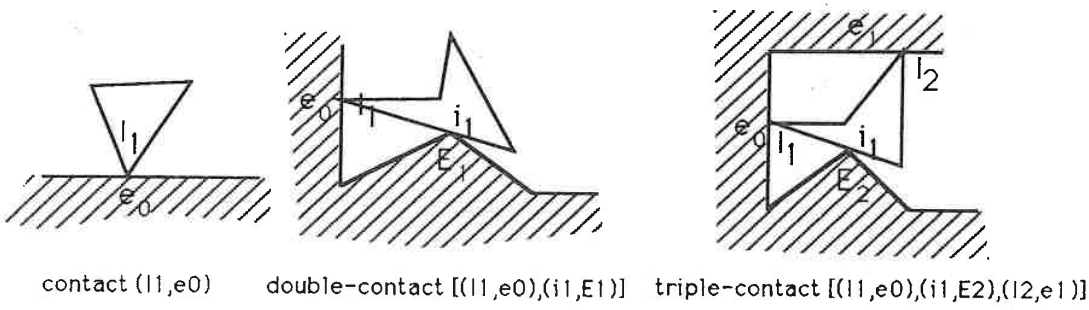


Figure 1 : Contacts simples, doubles et triples

2.3 Placement en translation pure

2.3.1 La situation retenue

Nous allons calculer dans un premier temps l'ensemble des placements libres de I par rapport à E. Rappelons que I et E sont des ensembles de polygones quelconques. Il existe 3 cas de placements libres. Pour chaque composante connexe de I et E, il se peut que :

- La composante connexe i de I soit incluse dans la composante connexe j de E.
- La composante connexe j de E soit incluse dans la composante connexe i de I.
- Les 2 composantes sont disjointes.

Ces 3 cas sont visualisés sur la figure 2.

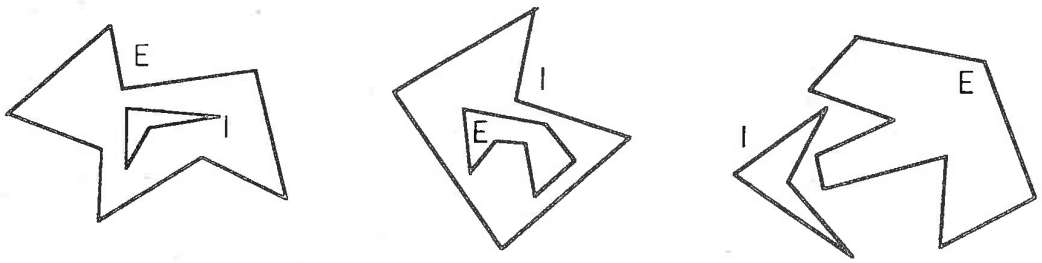


Figure 2 : Les 3 cas de placement libre

L'algorithme que nous allons présenter résoud ces 3 types de placements.
Nous allons nous restreindre dans la suite à une situation réaliste en robotique :
- La région polygonale E est composée d'un polygone PE_1 englobant un ensemble de polygones $PE_2, PE_3, \ldots, PE_l$.
- La région polygonale I est composée d'un seul polygone simple.

Un placement de I parmi E sera dit correct ou libre si :
- $I \subset PE_1$

• $I \cap PE_2 = \varnothing$,...,$I \cap PE_1 = \varnothing$

La figure 3 donne un exemple d'un tel placement.

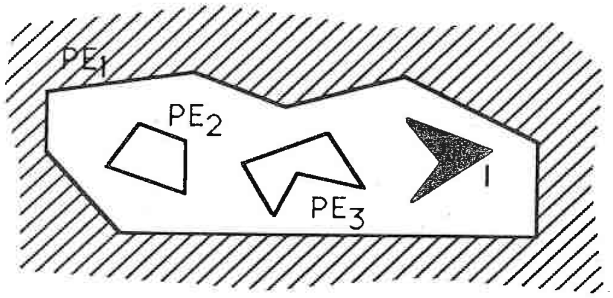


Figure 3 : Placement libre de I dans son environnement

Concrètement, PE_1 peut représenter le contour d'une pièce d'un appartement et PE_1,...,PE_l des meubles par exemple.

2.3.2 Allure de l'ensemble des placements

Soit xOy un repère fixe associé à E.

Soit I_0 un sommet particulier de I servant de point de référence. Un placement de I est complètement déterminé par la connaissance de la position de I_0 au sein du repère xOy. Soit α l'angle repérant la rotation de I considérée.

Notons $P(I,E,\alpha)$ l'ensemble des façons de placer sans collision en translation pure la rotation α de I parmi E.

On sait qu'il existe une bijection entre les placements en translation pure et $\mathbb{R}^2$. A tout point (x,y) de $\mathbb{R}^2$ on associe un placement et un seul. Il est obtenu en faisant coïncider le point considéré de $\mathbb{R}^2$ et le point de référence du polygone I. Il sera noté $I_{x,y}$.

Donnons maintenant la définition ensembliste de $P(I,E,\alpha)$:

$P(I,E,\alpha) = \{ (x',y') / I_{x',y'} \cap \complement PE_1 = \varnothing, I_{x',y'} \cap PE_2 = \varnothing, ..., I_{x',y'} \cap PE_l = \varnothing \}$ où $\complement PE_1$ désigne le complémentaire du polygone englobant PE_1 dans $\mathbb{R}^2$

On sait, pour le cas du placement en translation pure d'un polygone I parmi un ensemble de polygones E, que l'espace des configurations interdites vaut la différence de Minkowski entre I et E. L'espace libre $P(I,E,\alpha)$ valant bien sûr le complémentaire de l'espace des configurations interdites.

Cette différence est habituellement notée $E \ominus I$. Rappelons que c'est l'ensemble des x-y, x appartient à E, y appartient à I ramené à l'origine, où x-y désigne la différence vectorielle habituelle. On sait que l'allure de la différence de Minkowsky de 2 polygones convexes est un polygone convexe. Un polygone quelconque pouvant être décomposé en une union de polygones convexes, l'allure de $E \ominus I$ quand E et I sont quelconques, est celle d'un polygone quelconque

éventuellement non connexe donc P(I,E,α) est un polygone quelconque éventuellement non connexe. Dans la suite, nous nous intéresserons uniquement au bord de P(I,E,α) (noté @P(I,E,α)) ; P(I,E,α) pouvant être simplement déduit de @P(I,E,α).

Caractérisation des arcs et des sommets de @P(I,E,α)

Les arcs représentent des translations dues :
- à un sommet de I en contact avec un arc de E. On parlera de vertex-edge contact. Il sera noté (I_I,e_S)
- à un sommet de E en contact avec un arc de I. On parlera de edge-vertex contact. Il sera noté (i_I,E_S)

Les arcs dus à un vertex-edge contact sont parallèles à l'arc de E en question et les arcs dus à un edge-vertex contact sont parallèles à l'arc de I en question.
Un arc peut donc être labellé par un certain contact. Les sommets de @P(I,E,α) sont simplement les origines et les extrémités des arcs de @P(I,E,α). Un tel sommet peut être labellé par les contacts des 2 segments adjacents. Autrement dit un sommet de @P(I,E,α) peut être labellé par un certain double-contact.

Conclusion @P(I,E,α) est une région polygonale dont les arêtes sont des translations labellées par des contacts simples et dont les sommets sont des double-contacts. La figure 13 donne un exemple d'espace libre (visualisé en gris). Les arêtes de la zone grisée sont donc des translations correpondant à des contacts simples et les sommets sont des double-contacts.

2.4 Placement en translation et rotation

2.4.1 Introduction

Appelons P(I,E) l'ensemble des façons de placer I parmi E et @P(I,E) le bord de cet ensemble, (pour le cas d'étude que nous avons retenu, P(I,E) est un ensemble forcément borné puisque I doit être inclus dans PE_1).

On a immédiatement : $P(I,E) = \bigcup_{\alpha} P(I,E,\alpha)$, $\alpha \in [0,2\pi[$

P(I,E) est un sous-ensemble borné de $\mathbb{R}^2 X [0,2\pi[$. On a $(x,y,z) \in P(I,E) \Leftrightarrow (x,y) \in P(I,E,z)$.
Il devient assez difficile de se représenter mentalement l'allure de P(I,E). Des exemples en perspective sont donnés dans [A & B 88]. P(I,E) peut être connexe ou non. Rappelons que si les configurations de départ et d'arrivée n'appartiennent pas à la même composante connexe alors le déplacement n'est pas possible.

2.4.2 Etude de la variation en α de @P(I,E,α)

La méthode que nous allons développer s'inscrit dans le cadre général des méthodes de projections. Le paramètre à fixer sera bien entendu α. Nous connaissons la topologie d'une tranche de l'espace libre à α fixé. Rappelons qu'une telle tranche a l'allure d'un ensemble de polygones quelconques dont les sommets sont des double-contacts et dont les arêtes représentent des translations labellées par des contacts simples. Le but est donc de déterminer les α pour lesquels cette topologie change de façon discontinue. Certains sommets disparaissent tandis que d'autres apparaissent donnant naissance à de nouveaux arcs. Pour ce faire, nous allons nous intéresser à un double-contact particulier et déterminer les intervalles de $[0,2\pi[$ sur lesquels il est libre.

Définition Soit (c,c') un double-contact (c et c' sont des contacts simples). Nous appellerons $Df_{(c,c')}$ l'ensemble des intervalles de $[0,2\pi[$ sur lesquels le double-contact (c,c') est libre. Un intervalle sera dit intervalle de liberté s'il n'y a pas intersection entre I et E dans cet intervalle.

Remarque Nous étendons la notion de double-contact. Le contact c invoque un sommet et un arc. Nous considèrerons désormais que l'arc est remplacé par sa droite support. Le contact (i_l,E_k) est réalisé quand E_k appartient à la droite support de i_l. Nous parlerons dans ce cas de double-contacts généralisés.

Courbe décrite par un sommet de I pendant un double-contact généralisé
Soit (c,c') le double-contact auquel est contraint le polygone I et soit I_k le sommet de I dont on cherche à calculer le lieu des points parcourus. Nous cherchons l'équation dans le repère xOy attaché à E.

Règle Pour un certain α, le sommet I_k se trouve à l'intersection de la droite parcourue par I_k quand I est en translation suivant le premier contact et celle parcourue par I_k quand I est en translation suivnant le second contact (voir figure 4).

Equation de la droite décrite par I_k en translation suivant le contact (c,c')

Cas : (c,c') est un vertex-edge contact

Soit E_{j+1} le sommet suivant dans l'ordre CCW de E_j (CCW signifiant dans l'ordre contraire des aiguilles d'une montre).
Soit (E_j,E_{j+1}) la droite passant par E_j et E_{j+1} sur laquelle s'effectue le contact.
Soit i le rang du sommet de I concerné dans le contact.

On notera X.x l'abcisse du point X et X.y son ordonnée.
Soit Or.x et Or.y (resp. Fin.x et Fin.y) le point occupé par I_k lorsque le contact est en E_j (resp. E_{j+1})

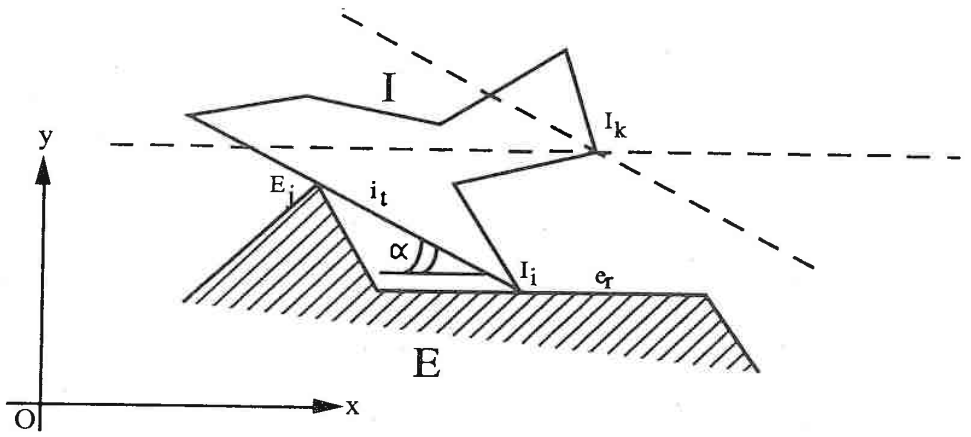


Figure 4 : Position de I_k pour le double-contact $((I_i,e_r),(i_t,E_j))$ et pour la rotation α de I

Soit $D_{i,\alpha}.x$ la différence en x pour la rotation α, entre le sommet origine I_0 et le sommet i. Pareillement $D_{i,\alpha}.y$ est défini comme la différence en y pour la rotation α, entre le sommet origine I_0 et le sommet i. On a :

$$Or.x = E_j.x - D_{i,\alpha}.x + D_{k,\alpha}.x$$
$$Or.y = E_j.y - D_{i,\alpha}.y + D_{k,\alpha}.y \quad (1)$$

$$Fin.x = E_{j+1}.x - D_{i,\alpha}.x + D_{k,\alpha}.x$$
$$Fin.y = E_{j+1}.y - D_{i,\alpha}.y + D_{k,\alpha}.y \quad (2)$$

Cas : (c,c') est un edge-vertex contact

Soit E_j le sommet de E en question dans le contact. Soit (I_i,I_{i+1}) la droite support de l'arc $[I_i,I_{i+1}]$ de I sur laquelle s'effectue le contact. Soit Or.x et Or.y (resp. Fin.x et Fin.y) le point occupé par I_k lorsque le contact est en I_i (resp. I_{i+1}). On a :

$$Or.x = E_j.x - D_{i,\alpha}.x + D_{k,\alpha}.x$$
$$Or.y = E_j.y - D_{i,\alpha}.y + D_{k,\alpha}.y \quad (3)$$

$Fin.x = E_j.x - D_{i+1,\alpha}.x + D_{k,\alpha}.x$ (4)
$Fin.y = E_j.y - D_{i+1,\alpha}.y + D_{k,\alpha}.y$

Nous pouvons maintenant exprimer les $D_{i,\alpha}$: (voir figure 5)

$D_{i,\alpha}.x = D_{i,0}.x * \cos(\alpha) - D_{i,0}.y * \sin(\alpha)$
$D_{i,\alpha}.y = D_{i,0}.y * \cos(\alpha) + D_{i,0}.x * \sin(\alpha)$

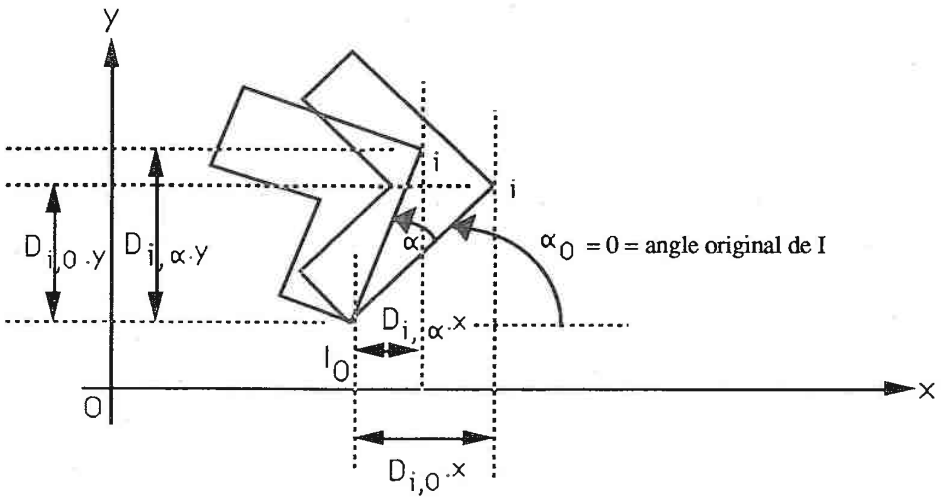


Figure 5 : Expression des $D_{i,\alpha}$

En remplaçant dans les expressions (1), (2), (3) et (4) nous avons :

Cas Vertex-edge contact :
$Or.x = a1.\cos(\alpha) + b1.\sin(\alpha) + c1$
$Or.y = a2.\cos(\alpha) + b2.\sin(\alpha) + c2$
$Fin.x = a1.\cos(\alpha) + b1.\sin(\alpha) + d1$
$Fin.y = a2.\cos(\alpha) + b2.\sin(\alpha) + d2$

Cas Edge-vertex contact :
$Or.x = a'1.\cos(\alpha) + b'1.\sin(\alpha) + c'1$
$Or.y = a'2.\cos(\alpha) + b'2.\sin(\alpha) + c'2$
$Fin.x = a'3.\cos(\alpha) + b'3.\sin(\alpha) + c'1$
$Fin.y = a4.\cos(\alpha) + b4.\sin(\alpha) + c'2$

3 cas se présentent suivant la nature du double-contact :

• Le double-contact est issu de 2 vertex-edge contacts. L'expression de l'intersection des 2 droites décrites par I_k donne :
$x(\alpha) = k_1*\cos(\alpha) + k_2*\sin(\alpha) + k_3$
$y(\alpha) = k_4*\cos(\alpha) + k_5*\sin(\alpha) + k_6$
Il s'agit ici d'une ellipse.

• Le double-contact est issu d'un vertex-edge contact et d'un edge-vertex contact. L'expression de l'intersection donne :

$x(\alpha) = (k_1*\cos^2(\alpha) + k_2*\sin^2(\alpha) + k_3*\sin(\alpha)*\cos(\alpha) + k_4*\sin(\alpha) + k_5*\cos(\alpha)) / (k_{11}*\cos(\alpha) + k_{12}*\sin(\alpha))$

$y(\alpha) = (k_6*\cos^2(\alpha) + k_7*\sin^2(\alpha) + k_8*\sin(\alpha)*\cos(\alpha) + k_9*\sin(\alpha) + k_{10}*\cos(\alpha)) / (k_{11}*\cos(\alpha) + k_{12}*\sin(\alpha))$

Il s'agit ici d'une courbe de degré 4.

• Le double-contact est issu de 2 edge-vertex contacts. L'expression de l'intersection donne :

$x(\alpha) = (k_1*\cos^3(\alpha) + k_2*\sin^3(\alpha) + k_3*\sin^2(\alpha)*\cos(\alpha) + k_4*\sin(\alpha)*\cos^2(\alpha) + k_5*\cos^2(\alpha) + k_6*\sin^2(\alpha) + k_7*\sin(\alpha)*\cos(\alpha)) / (k_{15}*\cos(\alpha) + k_{16}*\sin(\alpha))$

$y(\alpha) = (k_8*\cos^3(\alpha) + k_9*\sin^3(\alpha) + k_{10}*\sin^2(\alpha)*\cos(\alpha) + k_{11}*\sin(\alpha)*\cos^2(\alpha) + k_{12}*\cos^2(\alpha) + k_{13}*\sin^2(\alpha) + k_{14}*\sin(\alpha)*\cos(\alpha)) / (k_{15}*\cos(\alpha) + k_{16}*\sin(\alpha))$

Aprés simplification de ces 2 expressions, nous trouvons une forme du 4iéme degré.

Ces 3 cas sont illustrés sur la figure 6.

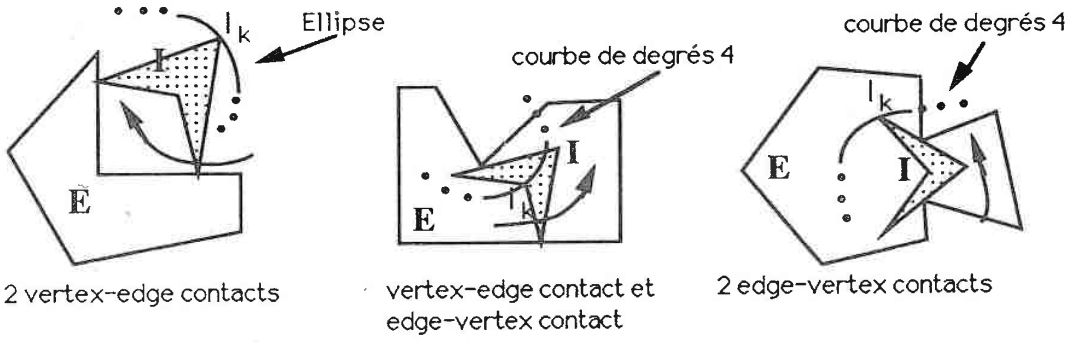


Figure 6 : Courbes décrites par I_k suivant les différents types de double-contact

La figure 7 donne un exemple de la courbe suivie par un sommet de I quand celui-ci est soumis à un double-contact de type vertex-edge,edge-vertex. On peut observer 4 branches infinies et vérifier que le degré est bien 4 (on ne peut pas placer une droite qui intersecte plus de 4 fois la courbe). Schwartz et Sharir analysent ces courbes dans [S & S 83].

Conclusion Nous avons réussi à établir les équations des courbes suivies par les sommets de I quand I est soumis à un double-contact. (3 types d'équation).
En considérant un repère attaché à E, certaines de ces courbes vont intersecter des arcs de E et symétriquement, en considérant un repère attaché à I, certaines des courbes suivies par les sommets de E vont intersecter des arcs de I. La connaissance de toutes ces intersections nous permet de calculer $Df_{(c,c')}$.

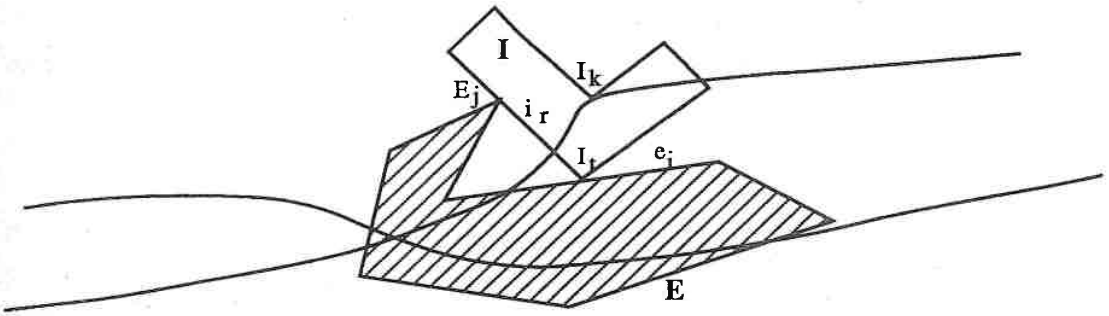


Figure 7 : Courbe suivie par I_k quand I est soumis au double-contact $((I_t,e_i),(i_r,E_j))$

Définition On considère comme positif le sens de rotation CCW. Soit (c,c') le double-contact auquel est soumis I.

On dira qu'un sommet de I est *entrant* sur un arc de E, pour une certaine rotation α de I, quand la courbe suivie par ce sommet intersecte l'arc de E et que pour la rotation $\alpha+d\alpha$, $\alpha \to 0$, le sommet de I est à l'intérieur de E.

On dira qu'un sommet de I est *sortant* sur un arc de E, pour une certaine rotation α de I, quand la courbe suivie par ce sommet intersecte l'arc de E et que pour la rotation $\alpha+d\alpha$, $\alpha \to 0$, le sommet de I est à l'extérieur de E.

Les notions d'entrance et de sortance valent aussi lorsque l'on considère les courbes suivies par les sommets de E dans un repère attaché à I.

La figure 8 illustre ces notions.

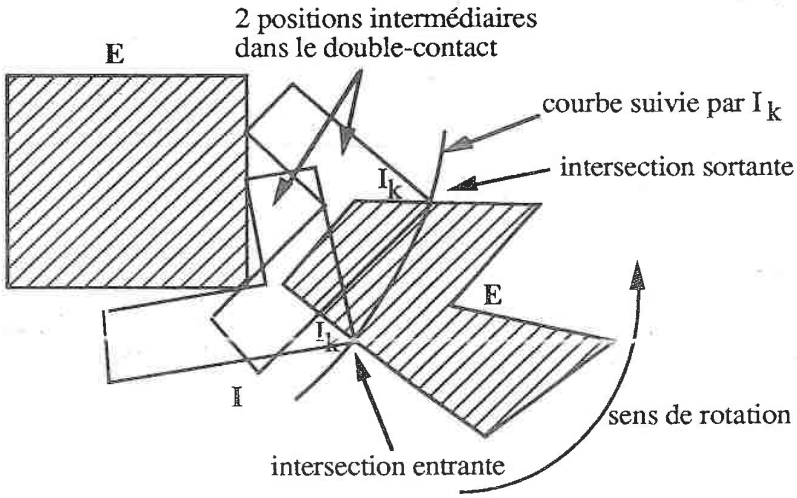


Figure 8 : Intersections entrantes et sortantes

Proposition 1 Pour un certain double-contact (c,c'), pour un certain sommet I_k de I, on peut déterminer en O(n) (n = nombre de sommets de E) les intersections entres les arcs de E et la courbe suivie par I_k. Symétriquement, pour un certain double-contact (c,c'), pour un certain sommet E_k de E, on peut déterminer en O(m) (m = nombre de sommets de I) les intersections entre les arcs de I et la courbe suivie par le sommet de E dans un repère lié à I. Chacune de ces intersections pourra être caractérisée comme entrante ou sortante.

Preuve La courbe suivie par le sommet de I ne peut intersecter un segment qu'un nombre fini de fois (ici 6 au maximum) vu le degré de cette courbe. Il peut donc y avoir au maximum 6n (resp. 6m) intersections entre la courbe suivie par I_k et les arcs de E (resp. intersections entre la courbe suivie par E_k et les arcs de I).

Proposition 2 Pour un certain double-contact (c,c'), on peut déterminer en O(m.n.log(m.n)) la liste triée en α croissant des intersections entre les courbes suivies par les sommets de I avec les arcs de E et celles suivies par les sommets de E avec I. La longueur de cette liste est en O(m.n).

Preuve Pour un certain double-contact (c,c'), les m sommets de I peuvent intersecter au maximum 6.m.n fois les arcs de E et les n sommets de E peuvent intersecter au maximum 6.m.n fois les arcs de I. Le tri coûtant bien sûr O(m.n.log(m.n)).

Définition On dira qu'il n'y a pas intersection entre I et E s'il n'y a pas d'intersection entre les arcs de I et ceux de E et s'il n'existe pas de sommet de I à l'intérieur d'un polygone de E ou vice-versa. La non-intersection peut donc se formaliser par :

- $\forall e_j \in E, \forall i_k \in I, e_j \cap i_k = \varnothing$
- $\forall I_k \in I, I_k \notin E$
- $\forall E_j \in E, E_j \notin I$

Définition Soit I_j un sommet de I, i_j un arc issu de I_j, e_i un arc de E. Supposons que I_j soit en contact avec e_i. On dira que i_j est sortant sur e_i avec contact I_j si i_j est dirigé vers l'extérieur de e_i et que i_j est entrant sur e_i avec contact I_j si i_j est dirigé vers l'intérieur de e_i (voir figure 9).

Algorithme de calcul de $Df_{(c,c')}$

Nous proposons un algorithme prenant en entrée la liste triée en α croissants de toutes les intersections. Il calcule en un passage sur cette liste la suite des intervalles de $[0,2\pi[$ pour lesquels il n'y a pas intersection entre I et E. Cette suite est éventuellement vide.

Structures de données

Les structures de données proposées renseignent à tout instant de l'appartenance ou de la non-appartenance d'un sommet de I à un polygone de E (et vice-versa) ainsi que de l'intersection de tout arc de I avec tout arc de E. On a :

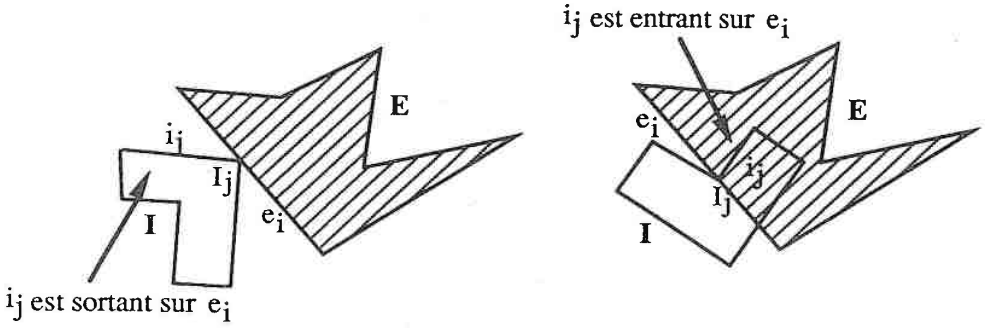


Figure 9 : Entrance et sortance d'arcs

- IV: Tab[0..m-1] de booléens
  IV[i] = vrai ssi le sommet i de I est à l'intérieur de E, faux sinon
- EV: Tab[0..n-1] de booléens
  EV[i] = vrai ssi le sommet i de E est à l'intérieur de I, faux sinon
- Edge_Inter Tab[0..n-1,0..m-1] de booléens
  Edge_Inter[i,j] = vrai ssi l'arc i de I intersecte l'arc j de E, faux sinon

On peut alors reformaliser la notion d'intersection de la façon suivante :

Intersection (I,E) $\Leftrightarrow$ ($\vee_{i=0,m-1}$ IV[i]) $\vee$ ($\vee_{j=0,n-1}$ EV[j]) $\vee$ ($\vee_{i=0,m-1}$ $\vee_{j=0,n-1}$ Edge_Inter[i,j])

où $\vee$ représente le ou logique. Cette expression uniquement composée de booléens modélise donc l'intersection de I et de E. En particulier, il n'y a pas intersection entre I et E si tous ses n+m+n.m booléens sont tous faux. A chaque pas de l'algorithme (c'est-à-dire à chaque examen d'une intersection), il conviendra de comptabiliser le nombre de booléens faux afin de pouvoir conclure à l'intersection ou à la non-intersection de I et de E. Appelons Nb_Inter ce nombre de booléens.
Il sera donc possible à chaque pas de déterminer en temps constant s'il y a ou non intersection. 4 cas se présentent :

- l'intervalle courant est un intervalle de liberté et il n'y a pas intersection entre I et E : on prolonge l'intervalle courant.
- l'intervalle courant est un intervalle de liberté et il y a intersection entre I et E. L'intervalle courant est comptabilisé et on crée un nouvel intervalle courant comme étant un intervalle de non-liberté.
- les 2 autres cas sont symétriques des 2 précédents.

On supposera que la liste triée en entrée est formée d'éléments structurés du type suivant :

```
type evt =   Record
             Angle      integer
             IE         boolean
             Sommet     integer
             Arc        integer
             Genre      boolean
             end;
```

où: Angle = la rotation pour laquelle se produit l'intersection

IE = vrai s'il s'agit d'un sommet de I entrant ou sortant d'un arc de E, faux s'il s'agit d'un sommet de I entrant ou sortant d'un arc de I.

Sommet = de quel sommet de I ou E il s'agit.

Arc = de quel arc de I ou E il s'agit.

Genre = vrai si le sommet est entrant sur un arc, faux sinon.

Un élément de cette liste sera appelé évènement.

Algorithme de calcul des intervalles de liberté du double-contact (c,c')

Initialisations

Les structures de données IV, EV, Edge_Inter doivent être initialisées pour une certaine rotation de I. Rappelons que tout l'intervalle [0,2π[est examiné et que pour tout α il existe un positionnement unique de I en double-contact (c,c') avec E. Nous choisissons arbitrairement la rotation α=0 pour initialiser les structures de données. Il s'agit de calculer les intersections entre les arcs de I (pour ce positionnement) et les arcs de E ainsi que les inclusions de sommets de I dans E et vice-versa. A l'issue de l'initialisation, on connait Nb_Inter, le nombre de booléens à faux.

Pour évènement dans liste répéter

*cas : **évènement.IE = vrai et évènement Genre = vrai*** *(voir figure 10 a et b)*

/ sommet de I entrant sur un arc de E */*

$I_j \leftarrow$ *évènement.Sommet*

$i_j, i_{j-1} \leftarrow$ *arcs issus du sommet* I_j

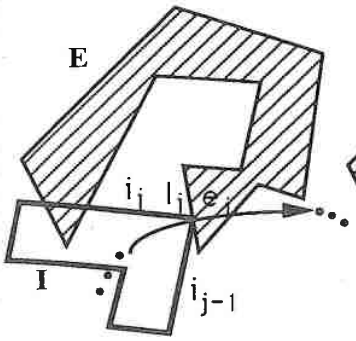
$e_i \leftarrow$ *évènement.Arc*

$IV[I_j] \leftarrow$ *vrai /* le sommet* I_j *est dans E */*

Nb_Inter = Nb_Inter-1

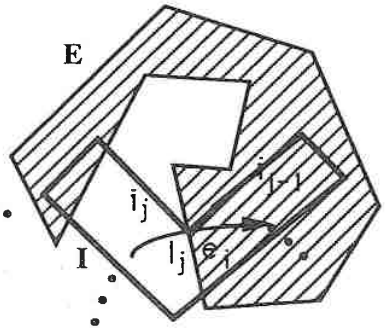
si Sortant(i_j, e_i, I_j)

```
                alors   Edge_Inter[ij,ei] ← vrai
                        Nb_Inter = Nb_Inter-1
                sinon   Edge_Inter[ij,ei] ← faux
                        Nb_Inter = Nb_Inter+1
        si Sortant(ij-1,ei,Ij)
                alors   Edge_Inter[ij-1,ei] ← vrai
                        Nb_Inter = Nb_Inter-1
                sinon   Edge_Inter[ij-1,ei] ← faux
                        Nb_Inter = Nb_Inter+1
        Mettre à jour la liste des intervalles de liberté suivant la valeur de Nb_Inter
    cas : évènement.IE = vrai et évènement Genre = faux (voir figure 10 c)
        /* sommet de I sortant d' un arc de E */
        Ij ← évènement.Sommet
        ij,ij-1 ← arcs issus du sommet Ij
        ei ← évènement.Arc
        IV[Ij] ← faux  /* le sommet Ij n'est plus dans E */
        Nb_Inter = Nb_Inter-1
        si Sortant(ij,ei,Ij)
                alors   Edge_Inter[ij,ei] ← faux
                        Nb_Inter = Nb_Inter-1
                sinon   Edge_Inter[ij,ei] ← vrai
                        Nb_Inter = Nb_Inter+1
        si Sortant(ij-1,ei,Ij)
                alors   Edge_Inter[ij-1,ei] ← faux
                        Nb_Inter = Nb_Inter-1
                sinon   Edge_Inter[ij-1,ei] ← vrai
                        Nb_Inter = Nb_Inter+1
        Mettre à jour la liste des intervalles de liberté suivant la valeur de Nb_Inter
    cas : évènement.Quel_Pol = faux et évènement Genre = vrai
        /* sommet de E entrant sur un arc de I */
        /* même raisonnement qu'avec le premier cas */
    cas : évènement.Quel_Pol = faux et évènement Genre = faux
        /* sommet de E sortant d' un arc de I */
        /* même raisonnement qu'avec le second cas */
    fincas
fin pour
Fin
```



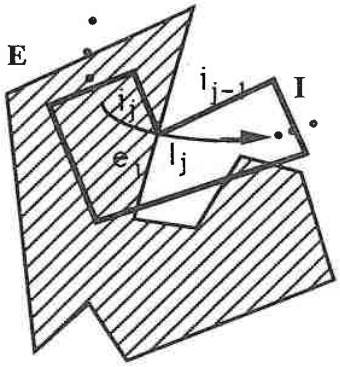
Il va y avoir intersection entre l'arc ij de I et l'arc ei de E car ij est sortant sur ei. Idem pour l'arc ij-1 de I.

Figure 10 a



Il ne va plus y avoir intersection entre l'arc ij-1 de I et l'arc ei de E car ij-1 est entrant sur ei. Cas contraire pour l'arc ij de I.

Figure 10 b



Il va y avoir intersection entre l'arc ij de I et l'arc ei de E car ij est entrant sur ei. Il ne va plus y avoir intersection entre l'arc ij-1 et ei car ij est sortant sur ei.

Figure 10 c

Remarque La liste des intervalles de validité tient compte de la notion de double-contact généralisé. Il convient de réduire cette liste aux seuls angles pour lesquels I a effectivement 2 contacts avec E. Cette opération très simple peut se faire en un passage sur la liste des intervalles de liberté.

Complexité de l'algorithme et conclusion

Cet algorithme calcule les intervalles de liberté d'un certain double-contact. La liste en entrée est triée (coût en O(m.n.log(m.n))) et est de longueur O(m.n). Le coût de l'initialisation est en O((m+n+K).log(n+m)) où K est le nombre d'intersections entre les arcs de I et ceux de E. Nous pouvons négliger ce terme devant le coût du tri. Pour un certain évènement le traitement se fait en temps constant donc le coût de l'algorithme est en O(m.n.log(m.n)). Il y a $m^2.n^2$ double-contacts possibles donc la détermination de tous les intervalles de liberté de tous les double-contacts peut se faire en $O(m^3.n^3.\log(m.n))$ dans le pire cas. C'est le mieux qu'on sache faire actuellement (à notre connaissance).

2.4.3 Calcul de @P(I,E,α)

La connaissance de tous les intervalles de liberté de tous les double-contacts nous permet de calculer de façon exacte le bord de @P(I,E,α). La liste des intervalles de validité d'un double-contact est de longueur O(m.n). Pour un α donné, savoir si un certain double-contact est valide ou pas peut se faire en log(m.n) vu l'ordonnancement en α de cette liste. Donc la détermination, pour un α donné, de tous les double-contacts valides peut se faire en $m^2.n^2.\log(m.n)$ (voir la figure 11).

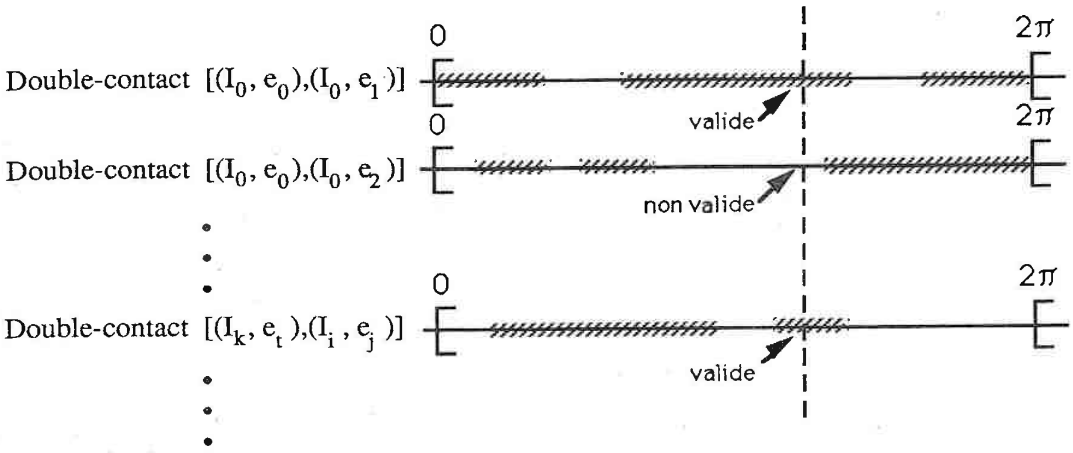
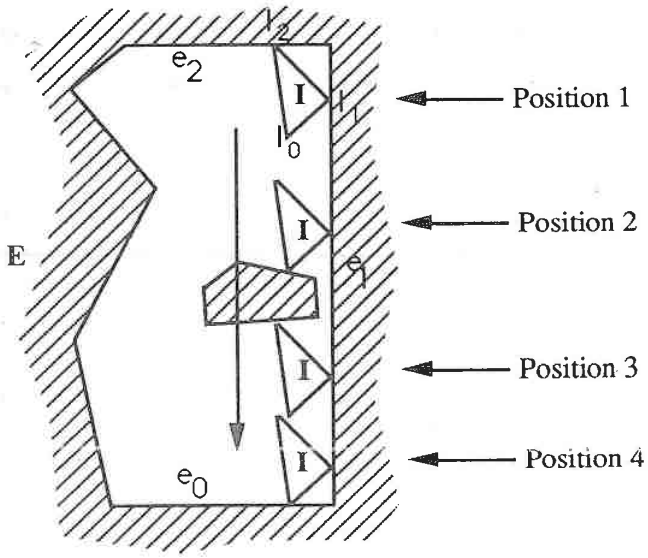


Figure 11 : Détermination des double-contacts libres pour un α donné

Appelons L_Vad(α) la liste des double-contacts libres pour la rotation α de I. Cette liste permet, après un pré-traitement de construire @P(I,E,α). Rappelons que @P(I,E,α) a l'allure d'un ensemble de polygones et que ses sommets sont des double-contacts. Les sommets de @P(I,E,α) sont donc contenus dans L_Vad(α) et il reste à les ordonner pour obtenir les contours polygonaux.

Définition Soit $L_{Ci}(\alpha)$ une liste de double-contacts contenant tous les contacts simples Ci. L'ordonnancement géométrique de $L_{Ci}(\alpha)$ consiste à ordonner les double-contacts de $L_{Ci}(\alpha)$ le long du segment intervenant dans le contact Ci (voir figure 12).



Considérons le contact simple (I1,e1). Il y a 4 double-contacts qui contiennent ce contact simple. Un tri permet de les ordonner le long de e1. On obtient ainsi les positions 1,2,3 et 4.

Figure 12 : Ordonnancement géométrique de double-contacts

Sur cet exemple, le suivant géométrique du double-contact en position 1 est celui en position 2.

Prétraitements sur L_Vad(α) et complexité

La longueur de L_Vad(α) est en $O(m^2.n^2)$ car il y a au plus $m^2.n^2$ double-contacts. Supposons que L_Vad(α) soit de la forme $((C_{1,1}, C_{1,2}), (C_{2,1}, C_{2,2}), \ldots, (C_{k,1}, C_{k,2}))$ où $(C_{i,1}, C_{i,2})$ est le ième double-contact composé des 2 contacts simples $C_{i,1}$ et $C_{i,2}$.

• Soit L_Vad1(α) la liste obtenue en triant L_Vad(α) suivant les $C_{i,1}$, i=1..k puis en ordonnant géométriquement chaque sous-liste contenant le même $C_{i,1}$ (il s'agit de l'ordonnancement géométrique des double-contacts ayant même $C_{i,1}$ le long du segment intervenant dans le contact simple $C_{i,1}$).

• Soit L_Vad2(α) la liste obtenue en triant L_Vad(α) suivant les $C_{i,2}$, i=1..k.puis en ordonnant géométriquement chaque sous-liste contenant le même $C_{i,2}$. Il reste encore à permuter les $C_{i,1}$ et les $C_{i,2}$ (de manière à présenter la clé en première position).

Le coût de l'obtention de L_Vad1(α) et L_Vad2(α).est en $O(m^2.n^2.\log(m.n))$.

• Soit L_Vad_Geo(α) la liste obtenue en interclassant L_Vad1(α) et L_Vad2(α) en respectant l'ordonnancement géométrique. Cet interclassement coûte $O(m^2.n^2)$. L_Vad_Geo(α) permet pour un certain double-contact (C1,C2) de déterminer son suivant géométrique. Cette détermination se fait en 2 temps :

• Localisation dans L_Vad_Geo(α) de la sous-liste contenant tous les double-contacts commençant par C1. Celle-ci se fait en log(m.n) et la sous-liste est de longueur O(m.n).

• Localisation dans la sous-liste (ordonnée géométriquement) du double-contact recherché puis rendu de son suivant dans la sous-liste. Celle-ci se fait encore en log(m.n).

Soit $L\_Vad_{Ci}(\alpha)$ la liste des double-contacts de L_Vad(α) contenant le contact Ci. Les 2 listes L_Vad1(α) et L_Vad2(α) permettent d'obtenir $L\_Vad_{Ci}(\alpha)$ en O(log(m.n)) opérations et la longueur de $L\_Vad_{Ci}(\alpha)$ est en O(m.n) (utilisation de 2 pointeurs sur L_Vad1(α) et L_Vad2(α)). Il reste encore à ordonner $L\_Vad_{Ci}(\alpha)$ "géométriquement" le long du segment relatif au contact Ci.

Conclusion L'obtention de L_Vad_Geo(α) coûte $O(m^2.n^2.\log(m.n))$ et la recherche d'un suivant géométrique coûte log(m.n).

Algorithme de calcul de @P(I,E,α)

Déroulement à la main de l'algorithme sur l'exemple de la figure 13 (idée d'un jeu de dominos).

Prenons comme double-contact de départ le double-contact $[(I_5,e_3),(I_4,e_2)]$. Recherchons le double-contact faisant intervenir le contact (I_4,e_2). Il s'agit de $[(I_4,e_2),(I_3,e_1)]$. Cette recherche se fait en O(log(m.n)). On crée l'arc reliant les 2 positions du sommet origine de I ; ce qui correspond à l'arc $[M_1,M_2]$ de l'exemple. Le double-contact courant devient le double-contact $[(I_4,e_2),(I_3,e_1)]$. Recherchons le double-contact faisant intervenir le contact (I_3,e_1). Il s'agit de $[(I_3,e_1),(i_3,E_1)]$. On crée alors l'arc correspondant à $[M_2,M_3]$, etc... Si la recherche rend une liste de double-contacts alors il faudra retenir le suivant géométrique du double-contact. Cette opération se faisant aussi en log(m.n) n'agrave pas la complexité.

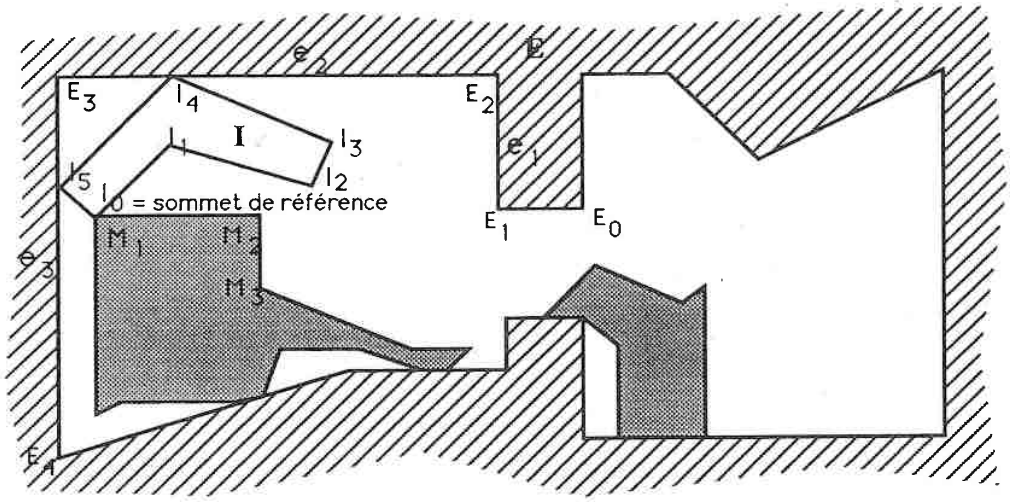


Figure 13 : Construction de @P(I,E,α)

Complexité de l'algorithme de construction de @P(I,E,α)

On sait donc trouver en O(log(m.n)) le double-contact faisant intervenir un certain contact. La complexité de l'algorithme est donc O(T*log(m.n)) où T est le nombre de sommets de @P(I,E,α). Dans le pire cas T vaut $m^2.n^2$ puisqu'il y a au plus $m^2.n^2$ double-contacts. La complexité dans le pire cas est donc en $O(m^2.n^2.\log(m.n))$. Rappelons que la construction de @P(I,E,α) nécessite la connaissance de tous les intervalles de liberté de tous les double-contacts. Ce "prétraitemant" est en $O(m^3.n^3.\log(m.n))$ mais n'intervient qu'une fois pour toutes.

Algorithme

en entrée :
- L_Vad(α) la liste des double-contacts valides pour α donné.
- L_Vad_Geo(α) la liste doublement triée des double-contacts de L_Vad(α)

en sortie :
- Une suite d'arcs reliant des double-contacts de L_Vad(α) et formant un ensemble de polygones.

Variables :
- D.C. le double-contact courant.
- D.C_Suiv le double-contact suivant du double-contact courant.
- Premier_D.C. = un double-contact quelconque de L_Vad(α)

Fonction de service: La fonction Suivant délivre le double-contact suivant de celui en entrée sur le contour de @P(I,E,α).

Début

Prétraitements sur L_Vad(α) pour obtenir L_Vad_Geo(α)

Tant que** L_Vad(α) non vide **répéter

D.C. ← *tête(L_Vad(α))*
D.C_Suiv ← *suivant(L_Vad_Geo(α),D.C.)*
Premier_D.C. ← *D.C.*
Creer_Arc(D.C.,D.C_Suiv)
Oter(L_Vad(α),D.C.)
Oter(L_Vad(α),D.C_Suiv)
D.C. ← *D.C_Suiv*
Tant que *D.C.≠ Premier_D.C.* ***répéter***
D.C_Suiv ← *suivant(L_Vad_Geo(α),D.C.)*
Creer_Arc(D.C.,D.C_Suiv)
Oter(L_Vad(α),D.C_Suiv)
D.C. ← *D.C_Suiv*
Fin Tant que
Fin Tant que

Conclusion Cet algorithme est particulièrement simple et donne en $O(m^2.n^2.\log(m.n))$ le bord de @P(I,E,α).

2.4.4 Calcul de @P(I,E)

Le principe de base de l'algorithme que nous allons présenter est simple. Rappelons que la méthode développée s'incrit dans le cadre général des méthodes de projection et que le paramètre fixé est α. Nous savons calculer une tranche de @P(I,E). Une tranche de @P(I,E,α) est composée d'un ensemble de polygones dont les sommets sont des double-contacts et dont les arcs sont des translations. Il suffit de déterminer pour quelles valeurs de α il y a modification de la tranche courante. Ces valeurs dites critiques de α sont simplement les débuts et les fins des intervalles de liberté des double-contacts. Elles correspondent à des triple-contacts (voir figure 14).

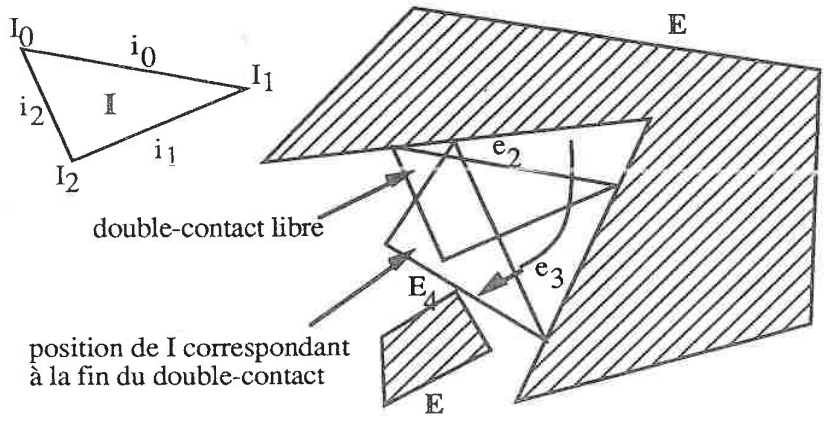
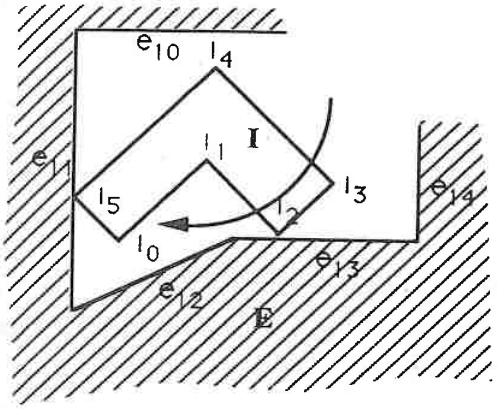


Figure 14 : Triple-contact marquant la fin (ou le début) d'un intervalle de liberté d'un double-contact

Sur l'exemple de la figure 14, l'intervalle de liberté du double-contact $[(I_0,e_2),(I_1,e_3)]$ prend fin pour la position correspondante au triple-contact $[(I_0,e_2),(I_1,e_3),(i_1,E_4)]$.
Déroulons l'algorithme à la main sur un premier exemple.



Soit le double-contact [(I2,e13),(I5,e11)] (noté D1). Il a pour voisins les double-contacts [(I4,e10), (I5,e11)] (noté D2) et [(I3,e14),(I2,e13)] (noté D3). Il existe donc des relations d'adjacence entre D1 et D2 ainsi qu'entre D1 et D3. D1 prend fin au triple-contact [(I5,e11),(I2,e12),(I2,e13)] (noté T1). Pour cet angle il existe 2 double-contacts qui débutent [(I5,e11),(I2,e12)] et [(I2,e12),(I2,e13)]. (notés D4 e D5). On doit alors :

- créer un sommet de type "extrémité" S et 2 sommets de type "intermédiaire" S2 et S3.
- dire que D1 se termine en S et dire que D4 et D5 commencent en S.
- dire qu'il existe une relation d'adjacence entre D4 et D5.
- dire qu'il existe une relation d'adjacence entre D2 et D4.
- dire qu'il existe une relation d'adjacence entre D3 et D5.

Cette étape de base peut être représentée graphiquement par le schéma de la figure 15. Les adjacences définissent des faces qui sont bordées par des arcs représentant soit des double-contacts (α variant) soit des translations (α fixé). Une face est telle que les équations de ses arcs gardent une forme analytique constante.

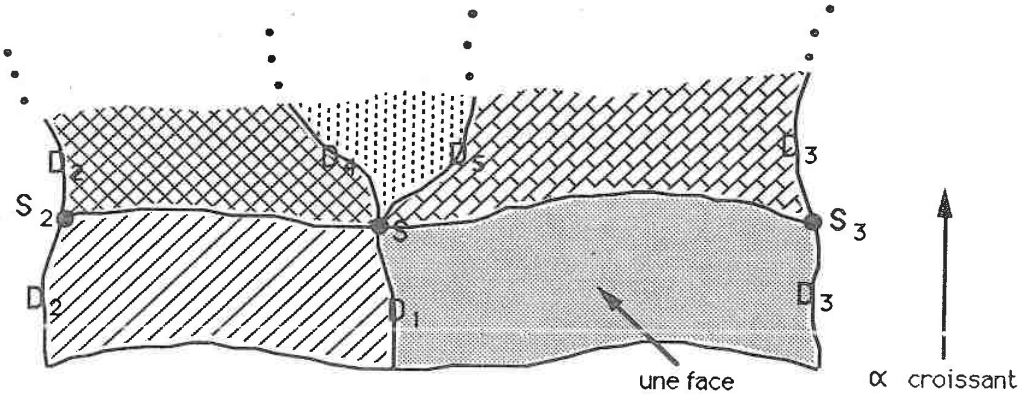
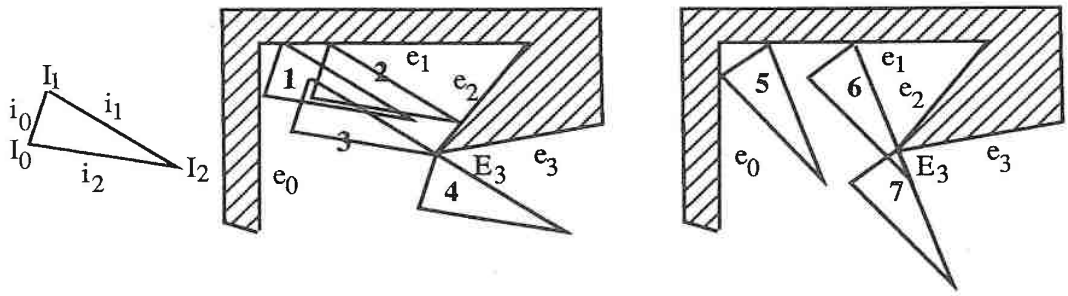


Figure 15 : Construction de @P(I,E)

Définition On dit qu'un sommet est de type extrémité lorsqu'il correspond à une fin ou à un début de double-contact. On dit qu'un sommet est de type intermédiaire quand il est placé sur un arc correspondant à un même double-contact.

Déroulement sur un second exemple :



Considérons la figure de gauche. Soient les double-contacts relatifs aux positions 1,2,3 et 4. Ils sont tels que 1 adjacent à 2, 2 adjacent à 3 et 3 adjacent à 4. Les double-contacts 2 et 3 prennent fin quand I2 coïncide avec E3 et quand I1 est en contact avec e1. Ces 2 double-contacts donnent naissance au double-contact relatif à la position 6 (sur la figure de droite). Ce double-contact a pour double-contacts adjacents les double-contacts relatifs aux positions 5 et 7 qui sont les mêmes que ceux des positions 1 et 4. Ceci peut se symboliser par le schéma donné en figure 16.

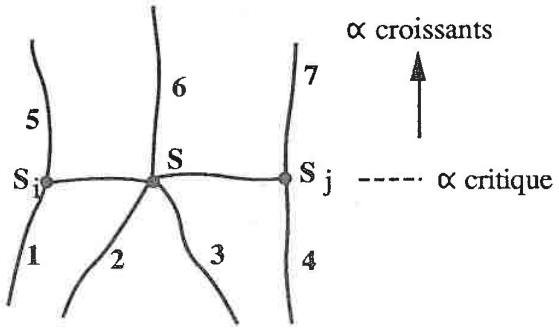


Figure 16 : Construction de @P(I,E)

Structures de données en jeu dans l'algorithme de construction de @P(I,E)

P(I,E) se présente comme un graphe tridimensionnel de sommets reliés par des arcs représentant soit des double-contacts soit des translations. Un sommet est l'origine ou l'extrémité de 3 types d'arcs :

- des arcs correspondant à des double-contacts commençant à ce sommet (∝ croissant).
- des arcs correspondant à des double-contacts arrivant à ce sommet (∝ décroissant)
- des arcs correspondant à des adjacences avec d'autres sommets (voir la figure 16 entre le sommet S et S_i ou bien S et S_j).

On peut alors proposer les structures et types suivants (notation en pseudo Pascal) :

```
• type Double-contact =
        Record
        C1,C2 : integer;                /* 2 entreés dans la structure A                */
        End;

• type Arc =
        Record
        DC : Double-contact;
        Extr : Pt_Sommet;               /* pointeur sur le sommet extrémité de l'arc    */
        End;

• type Sommet =
        Record
        X,Y : Real;                     /* coordonnées du point de référence de I       */
        Angle : Real;                   /* angle de rotation de I                       */
        Nb_Depart : Integer;            /* nombre d'arcs issus du sommet vers les α     */
                                        /* croissants                                   */
        Depart : array[1..n] of Arc;    /* liste des arcs issus du sommet en provenance */
                                        /* des α inférieurs                             */
        Nb_Arrivee : Integer;           /* nbre d'arcs arrivant au sommet
        Arrivee : array[1..n] of Arc;   /* liste des arcs arrivant au sommet            */
        Adj_G,Adj_D : Pt_Sommet;        /* pointeurs vers les sommets adjacents (au maxi-*/
                                        /* mum 2                                        */
        End;

• type Pt_Sommet = ^Sommet;             /* type pointeur sur sommet                     */

• A : array[1..n.m,1..n.m] of Pt_Sommet; /* A[i,j] = pointeur sur le sommet origine du   */
                                        /* double-contact (i,j)                         */

• PIE : Array[1..m3.n3] of Sommet;      /* liste des sommets de PIE                     */
• Nb_Sommet : Integer;                  /* nombre de sommets courants dans PIE          */
```

Algorithme

en entrée : • La liste L_Vad de tous les intervalles de validité de tous les double-contacts. Cette liste se présente comme une suite ordonnée en α d'évènements qui sont soit des débuts de double-contact soit des fins. On supposera qu'un évènement est de la forme <x,y,α,D/F,(C1,C2)> où :

- x,y = position du sommet origine de I
- α = orientation de I
- D/F = "debut" si l'évènement marque le début d'un double-contact, "fin" sinon.
- (C1,C2) = le double-contact lui même.

en sortie : • @P(I,E)

A l'étape courante, on aura :

• Retirer de la tête de L_Vad tous les éléments ayant même .x,.y,.α et constituer les 2 listes L_Debut et L_Fin telles que tous les éléments de L_Debut correspondent à des débuts d'intervalles de liberté et ceux de L_Fin à des fins d'intervalles de liberté.

• Créer un sommet S de type extrémité et mettre à jour les arcs relatifs aux double-contacts présents dans L_Fin. Ceci peut se traduire par :

• Initialiser un sommet S

• Pour chaque double-contact DC de L_Fin répéter

• Récupérer dans Or le sommet origine de l'arc qui doit se terminer en S

• Mettre à jour l'extrémité de l'arc de Or.Departs[] relatif au double-contact avec le sommet S (voir 1 à la figure 17).

• Créer un arc d'extrémité Or, de double-contact DC et l'ajouter dans la liste des arrivées au sommet S (voir 2 à la figure 17).

• Créer autant d'arcs qu'il y a d'éléments dans la liste L_Debut et les ajouter à la liste des Départs du sommet S. Ceci peut se traduire par :

• Pour chaque double-contact DC de L_Debut répéter

• Créer un arc d'extrémité indéfinie et de double-contact DC

• L'ajouter à la liste des Départs du sommet S (voir 3 à la figure 17).

• A[DC.C1,DC.C2] = pointeur sur le sommet S (Naissance d'un nouveau double-contact)

• Obtention de la liste L_Cont des double-contacts n'appartenant pas à L_Fin et possèdant des adjacences avec des double-contacts de L_Fin.Ceci revient à rechercher des double-contacts dans les arcs de départ des sommets adjacents aux sommets origines des arcs se terminant en S et conserver parmi ces double-contacts ceux partageant un contact avec ceux de L_Fin (voir 4 à la figure 17).

• Pour chaque élément de L_Cont, créer un sommet Si de type intermédiaire. Ceci peut se traduire par :

• Pour chaque double-contact DC de L_Cont répéter

• Initialiser un sommet Si

• Mise à jour avec Si de l'extrémité de l'arc relatif à DC dans les départs du sommet origine Or de cet arc (voir 5 à la figure 17).

• Créer un arc d'extrémité Or, de double-contact DC et l'ajouter aux arrivées de Si (voir 6 à la figure 17).

• Créer un arc de double-contact DC, d'extrémité indéfinie et l'ajouter aux Départs de Si (voir 7 à la figure 17).

• Créer 2 adjacences au sommet S avec les sommets créés à l'étape précédente (voir 8 à la figure 17).

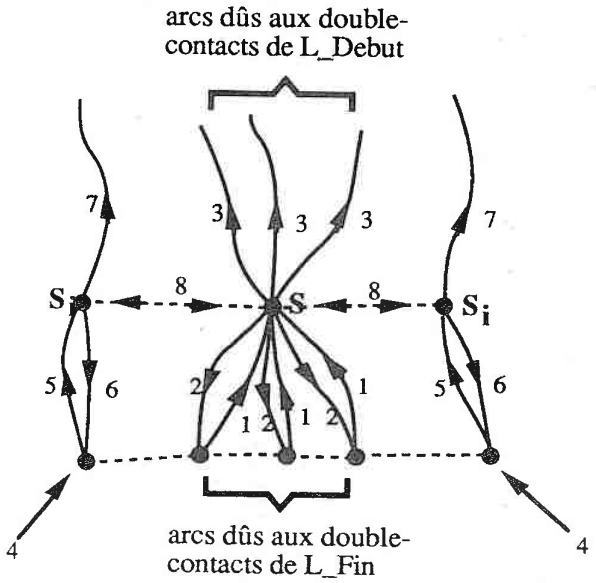


Figure 17 : Construction des arcs de P(I,E)

Complexité

L'algorithme itère sur la liste des triple-contacts présents dans L_Vad.et pour chacun d'eux effectue un traitement simple en temps constant. Il y a $O(m^3.n^3)$ triple-contacts donc l'algorithme a une complexité en $O(m^3.n^3)$. Le coût d'obtention de L_Vad étant de $O(m^3.n^3.\log(m.n))$ la complexité finale de l'algorithme de construction de @P(I,E) est aussi en $O(m^3.n^3.\log(m.n))$. C'est le mieux qu'on sache faire actuellement (à notre connaissance).

2.5 Application à la planification de trajectoires

Nous avons réussi à modèliser le bord de @P(I,E) par un graphe tridimensionnel de sommets reliés par des arcs. Il est déja possible de rechercher des trajectoires sur ce bord mais le modèle n'autorise pas actuellement la recherche de toutes les trajectoires. En effet, il se peut que le bord de

@P(I,E) ait plusieurs composantes connexes. Prenons l'exemple d'un polygone englobant rectangulaire et d'un obstacle de petite taille situé en son milieu. Si l'objet est suffisament petit pour que la composante de @P(I,E) liée à l'obstacle n'intersecte pas la composante de @P(I,E) liée au polygone englobant alors il ne sera pas possible de trouver un chemin entre une position en contact avec le polygone englobant et une position en contact avec l'obstacle. Intuitivement, il manque des "ponts" liant les 2 composantes de @P(I,E). Nous allons proposer une solution pratique pour créer de tels ponts.

Il suffit pour cela de considérer une tranche de l'espace libre pour une orientation fixée. Arbitrairement, nous prendrons alpha = 0°. S'il existe plusieurs composantes connexes de @P(I,E) qui peuvent communiquer, alors pour l'orientation 0, il sera possible de créer des arcs suplémentaires reliant les composantes. Remarquons dès maintenant qu'il est possible que toutes les composantes connexes ne puissent communiquer. Considérons 2 composantes qui ne peuvent communiquer. Ceci advient si l'une ne contient pas l'autre (composantes disjointes) ou, si l'une contient l'autre et s'il est impossible de créer un arc les reliant qui soit totalement dans l'espace libre.

La figure 18 indique, pour une certaine tranche de l'espace libre, les arcs qui peuvent être créés et ceux qui ne le peuvent pas.

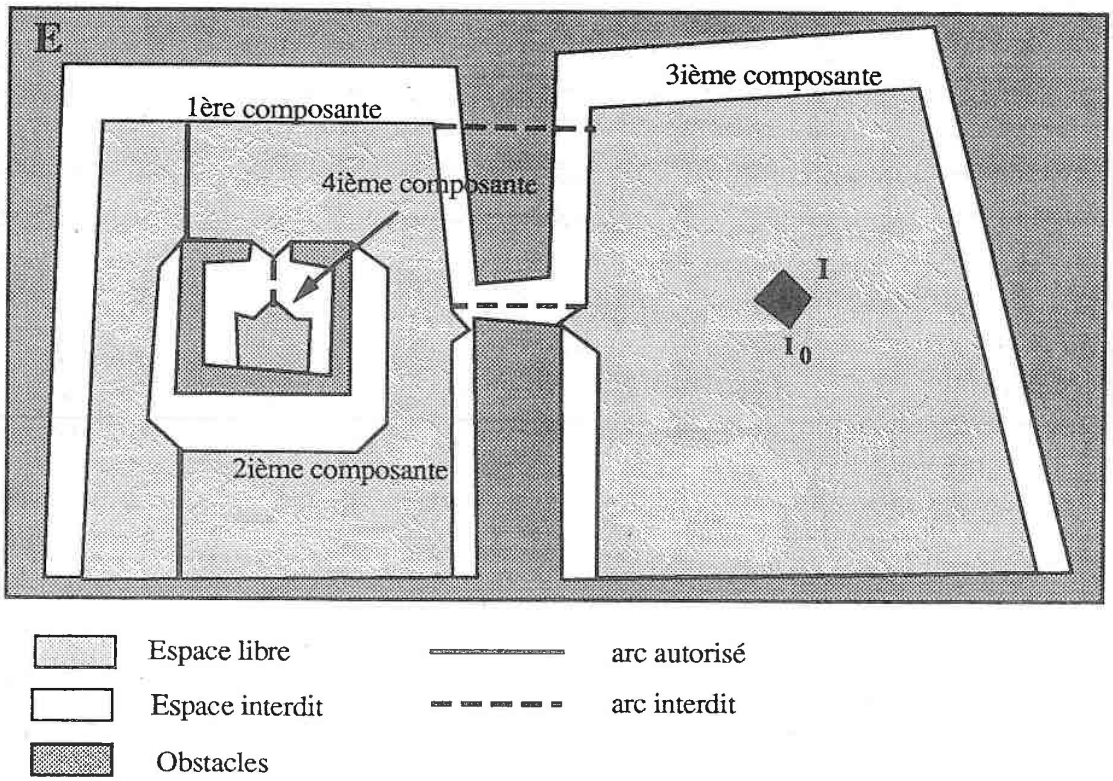


Figure 18 : Arcs autorisés et interdits pour les liaisons entre les différentes composantes connexes d'une tranche de l'espace libre

L'algorithme que nous allons proposer est de type Plane-sweep ([Meh 84]). Il prendra en entrée les sommets de @P(I,E,0) ainsi que ceux des obstacles. Cette liste de sommets (de longeur au plus $m^2.n^2$) sera triée en x croissants. A tout moment, l'algorithme maintiendra une structure de données fournissant la liste triée en y des segments intersectés par la sweep-line. L'examen du sommet courant provoquera suivant le cas une insertion ou une destruction de certains segments de la structure de données. Après chaque insertion ou destruction, certaines relations d'adjacence des segments de la structure de données sont modifiées. L'examen de celles-ci permettra, sous certaines conditions, de dire qu'il peut exister un nouvel arc entre les 2 segments considérés. les nouveaux arcs crées seront parallèles à la sweep-line (ce qui n'est pas le cas de tous ceux de la figure 18).
La condition à la création d'un nouvel arc est la suivante : il faut que les 2 segments adjacents considérés soient des segments appartenant à @P(I,E,0), qu'ils n'appartiennent pas à la même composante connexe et enfin que ce nouvel arc les reliant soit entiérement dans l'espace libre. Cette dernière sous-condition sera testée simplement en positionnant le robot I sur un point quelconque de l'arc et en déterminant s'il y a intersection entre I et E, ce test d'intersection coûtant O(m.n).
Donnons maintenant les structures de données et les fonctions de base de l'algorithme.

• Seg : Structure de données principale gérée par l'algorithme. Elle contient pour toute position de la sweep-line la liste triée en y des segments intersectés par celle-ci.

• Insérer(e_i,Seg,ea_1,ea_2) insère le segment e_i dans la structure Seg et rend dans ea_1 et ea_2 les segments voisins de e_i après l'insertion. Sa complexité est logarithmique en le nombre de segments présents dans Seg.

• Détruire(ei,Seg,ea_1,ea_2) détruit le segment e_i de la structure seg et rend dans ea_1 et ea_2 les segments voisins de e_i avant la destruction. Sa complexité est elle aussi logarithmique en le nombre de segments présents dans Seg.

• Num_Cp_Cnxe(ei) rend le numéro de la composante connexe à laquelle appartient e_i.

Algorithme

Pré-traitements
• Calcul de @P(I,E,0)
• Tri des sommets de @P(I,E,0) et de ceux de E en x croissants.
• Détermination des composantes connexes de @P(I,E,0) (pour chacun des sommets triés, on saura déterminer s'il s'agit d'un sommet de @P(I,E,0) ou de E et à quelle composante connexe il appartient).

Début

- Seg ← ∅
- Soit p le nombre de sommets triés
- Pour i de 1 à p répéter
    - /* traitement du sommet V_i*/
    - e_1,e_2 ← les 2 arcs issus du sommet V_i
    - Si e_1 est à droite de V_i

alors
- Insérer(e_1,Seg,ea_1,ea_2)
- [S1,S2] ← [Vi, intersection de la sweep-line avec ea_1]

(1) • Si $e_1 \in$ @P(I,E,0) et $ea_1 \in$ @P(I,E,0) et Num_Cp_Cnxe(e_1) ≠ Num_Cp_Cnxe(ea_1) et l'arc [S_1,S_2] est dans l'espace libre

alors • Créer un nouvel arc dans le modèle entre les sommets S_1 et S_2

- idem (1) avec ea_2

sinon
- Détruire(e_1,Seg,ea_1,ea_2)
- Idem (1) avec ea_1 et ea_2

Fin pour

Complexité et conclusion

L'itération principale comporte au plus $m^2.n^2$ boucles.
Les fonctions Insére et Détruit fonctionnent en O(log(m.n)). Le test permettant de déterminer si le nouvel arc est dans l'espace llibre ou pas coûte O(m.n).
La création d'un nouvel arc dans le modèle est une opération en 2 phases. L'arc créé est en fait composé de 2 arcs. Le premier relie S1 (qui correspond à un double-contact) à S2 (qui ne correspond à rien dans le modèle). Le second relie S2 à une extrémité quelconque de l'arc portant S2. Cette extrémité correspond à un double-contact. Il s'agit donc d'introduire sur certains arcs des sommets intermédiaires et à relier ces sommets. La localisation d'un certain arc dans le modèle coûte O(log(m.n)) vu le tri préalable réalisé sur tous les arcs du modèle.
La complexité finale est en $O(m^3.n^3)$ pour l'itération et en $O(m^3.n^3.\log(m.n))$ pour le tri des arcs du modèle. Cette mise à jour du modèle n'aggrave donc pas la complexité précédente.
La phase finale consiste simplement à rechercher une trajectoire dans le modèle. Elle est constituée de 3 phases :

- Soit $(X_{dep},Y_{dep},\alpha_{dep})$ la position de départ du polygone I. La première phase consiste à déterminer dans le modèle un sommet qui soit le plus proche possible de la position de départ. Pour cela, calculer @P(I,E,α_{dep}), déterminer le plus proche sommet S_p de @P(I,E,α_{dep}) depuis (X_{dep},Y_{dep}). Effectuer la translation de (X_{dep},Y_{dep}) à S_p puis réaliser le double-contact correspondant jusqu'au sommet de @P(I,E) marquant la fin du double-contact. Soit S_{dep} ce sommet.

• Répéter la phase 1 en considérant cette fois la position finale (X_{fin},Y_{fin},α_{fin}) mais sans réaliser les déplacements. Soit D_{fin} le sommet de @P(I,E) trouvé.
• Rechercher dans le modèle une "meilleure" trajectoire (pour un certain critère) entre les sommets S_{dep} et S_{fin}. Si cette trajectoire existe alors réaliser effectivement le déplacement.
• Réaliser le déplacement symétrique de celui effectué en phase 1 entre le sommet S_{fin} et (X_{fin},Y_{fin}).

On sait que la meilleure complexité de l'algorithme A* (recherche d'un plus court chemin , voir [TOU 88]) est en $O(m^3.n^3.\log(m.n))$ où $m^3.n^3$ est le nombre d'arcs et de noeuds du graphe.

2.6 Le logiciel développé

La difficulté principale de cette étude a été le calcul des intersections entre les courbes suivies par les sommets de I pendant un double-contact avec les arêtes de E. Les techniques d'ananlyse numériques se sont révélées délicates d'emploi au point que la méthode a été abandonnée. Nous avons developpé alors une version approchée de l'algorithme mais celle-ci suppose que si un déplacement est possible, alors les positions extrèmes du déplacement appartiennent à une même composante connexe de @P(I,E) (on suppose en effet que celles-ci appartiennent au bord de l'espace libre). Les résultats sont néanmoins spectaculaires et donnent une bonne idée de ce que devrait être l'algorithme présent s'il avait été effectivement programmé. Nous donnons quelques copies d'écran.en figures 19 et 20. Ce logiciel a été écrit en langage C.

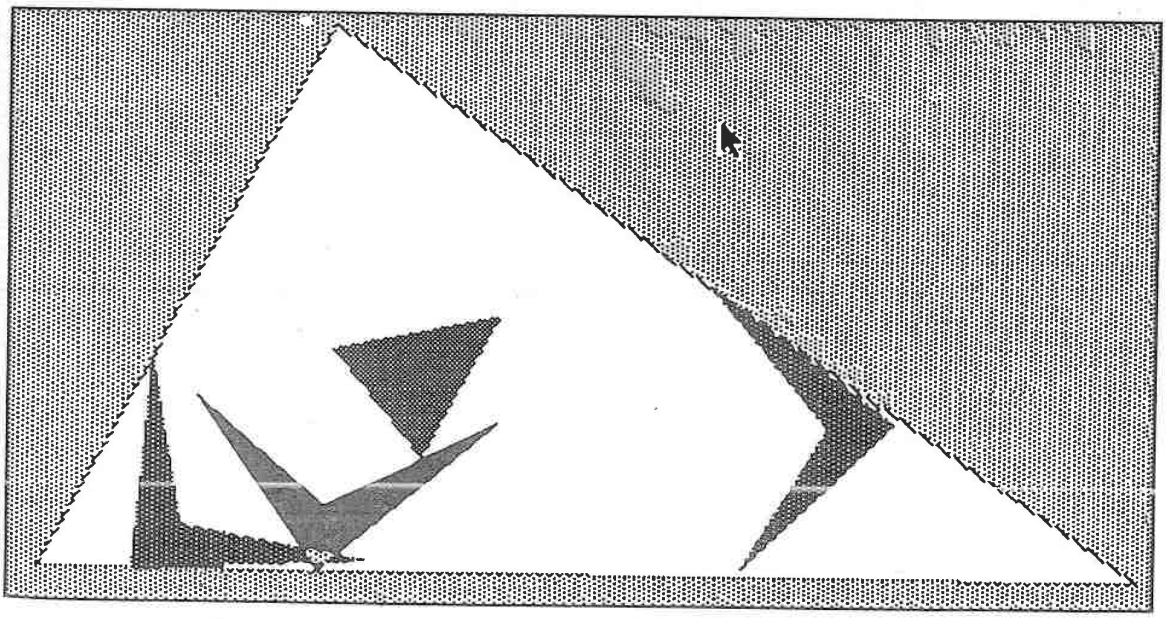


Figure 19 : Déplacement d'un polygone. Les positions de départ et d'arrivée figurent en gris foncé, les obstacles en gris clair et le polygone se déplaçant en noir.

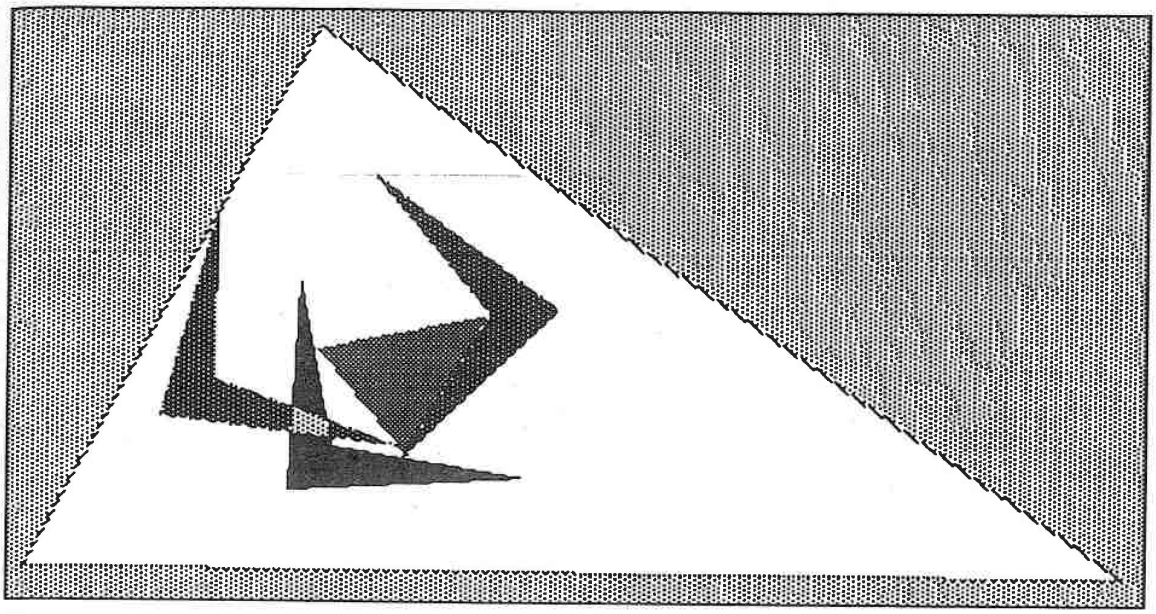


Figure 20 : Déplacement d'un polygone : mêmes conventions qu'à la figure précédente

2.7 Conclusion finale et perspectives

Nous avons proposé une solution exacte pour résoudre le problème du déplacement en translation et rotation d'un polygone au sein d'un ensemble de polygones. Nous avons tout d'abord créé un modèle du bord de l'espace libre puis nous avons étendu ce modèle à l'aide d'arcs en provenance de graphes des géodésiques d'un certain nombre de tranches de @P(I,E). La recherche d'une trajectoire se résume alors en la recherche d'un chemin dans un graphe.

Nous travaillons actuellement sur le déplacement non plus d'un polygone mais d'un splinegone. Les problèmes mathématiques y sont plus sérieux mais la structure générale de l'algorithme devrait pouvoir être conservée. Les notions de contact de type vertex-edge et edge-vertex sont étendues avec un 3ième type de contact ; celui entre 2 arcs puisque un arc de I (remplacé maintenant par une courbe) peut être en contact avec un arc de E (voir figure 19).

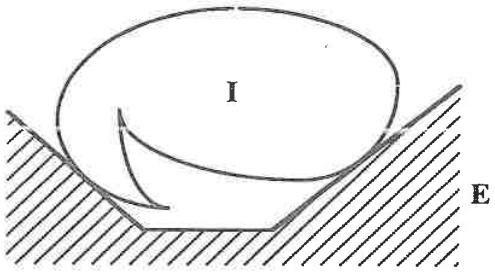


Figure 19 : Contacts entre un splinegone et un polygone

La principale difficulté de ce nouveau problème réside dans le fait qu'une arête de I (courbe) peut avoir une intersection avec une arête de E sans que les origines de l'arête de I pénètrent dans E ce qui complique singulièrement un des algorithmes vu précédement.

Chapitre 3

Diagrammes de Voronoï

3.1 Diagramme de Voronoï de points : définitions et algorithme

Le diagramme de Voronoï a toujours fait l'objet de beaucoup d'attention dans le milieu de l'algorithmique géométrique. Il est un outil remarquable dans tous les problèmes de localisation ainsi qu'en planification de trajectoires. Nous proposons dans ce chapitre plusieurs algorithmes de calcul du diagramme de Voronoï sur des objets divers. Nous allons donner tout d'abord quelques définitions générales ainsi que des propriétés.

Définition Soit S un ensemble de n points $p_1,p_2,...,p_n$ dans $\mathbb{R}^d$. Le diagramme de Voronoï de S est un pavage de $\mathbb{R}^d$ en n cellules $V(p_1),...,V(p_n)$ où $V(p_i)$ = ensemble des points de $\mathbb{R}^d$ situés plus près du point p_i que de tout autre point p_j, $j \neq i$.

Donnons une définition constructive du diagramme de Voronoï :

Définition Soit $h(p_i,p_j)$ l'ensemble des points de $\mathbb{R}^d$ situés plus près de p_i que de p_j. Cet ensemble est un demi-espace (un demi-plan dans $\mathbb{R}^2$). On a : $V(p_i) = \bigcap h(p_i,p_j)$ $i \neq j$. Cette méthode est celle suivie par l'humain lorsqu'il cherche à dessiner à la main le diagramme de quelques points dans le plan.

Autre définition :

Définition Soit A un point de $\mathbb{R}^d$. On appelle éloignement de A au sein des points p_i, la plus petite distance entre A et un point quelconque parmi les p_i. Le diagramme de Voronoï peut aussi être vu comme l'ensemble des points de $\mathbb{R}^d$ tels que pour chacun d'eux il n'existe pas de points de $\mathbb{R}^d$ plus éloignés des p_i et qui n'appartienne pas au diagramme.

Autre définition :

Définition Le diagramme de Voronoï de n points $p_1,p_2,...,p_n$ de $\mathbb{R}^d$ est l'ensemble des points A de $\mathbb{R}^d$ à égale distance d'au moins 2 points parmi les p_i et tels que ces p_i soient les plus proches de A.

Cette dernière définition est certainement celle qui s'apparente le mieux à la planification de trajectoires. Elle rend l'idée que le diagramme peut être utilisé comme une espèce de "route" au milieu des obstacles. Notons que certaines définitions définissent les frontières des cellules de Voronoï et que d'autres définissent les frontières et l'intérieur des cellules.

Le diagramme de Voronoï de n points dans la plan a l'allure d'un ensemble de polygones convexes disjoints dont l'union est tout le plan (voir figure 1). Dans $\mathbb{R}^3$, le diagramme est un ensemble de polytopes convexes.

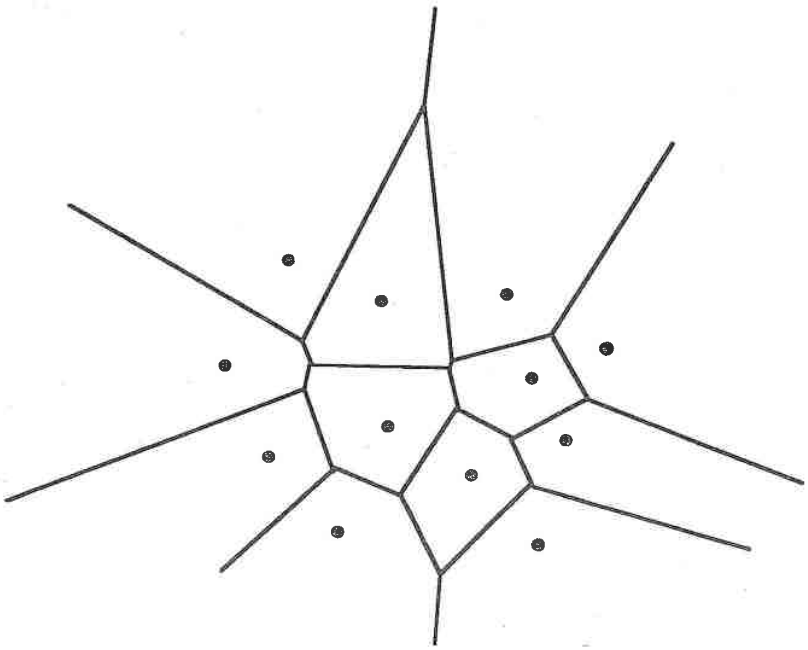


Figure 1 : Diagramme de Voronoï de points

Donnons ici quelques propiétés du diagramme de Voronoï

Propriétés du diagramme de Voronoï (voir [P & S 85])

• Si nous faisons la supposition qu'il n'y a pas 4 points cocirculaires alors chaque sommet du diagramme est l'intersection d'exactement 3 arcs du diagramme. Le mot arc est à prendre au sens large. Il est fonction de d (segment dans $\mathbb{R}^2$, facette dans $\mathbb{R}^3$).

• Chaque plus proche p_j du point p_i, $i \neq j$, définit un arc de $V(p_i)$.

• $V(p_i)$ est un polygone ouvert si et seulement si p_i est situé sur l'enveloppe convexe de l'ensemble des points.

• Soit v un sommet du diagramme et soit C(v) le cercle de centre v et passant par les 3 points p_i,p_j,p_k donnant naissance à v. Le cercle C(v) est vide de points.

• Le nombre d'arcs et de sommets du diagramme de Voronoï de n points est linéaire en n. Ce résultat est capital pour les calculs de complexité des algorithmes utilisant un diagramme de Voronoï.

Définition Soient S1 et S2 deux ensembles d'objets. On appellera bissectrice de S1 et S2 l'ensemble noté B(S1,S2) tel que pour chacun de ses points x, la plus petite distance de x à un point de S1 est égale à la plus petite distance de x à un point de S2.
Exemples :

- Si S1 et S2 sont 2 points du plan alors B(S1,S2) est la médiatrice de S1 et S2.
- Si S1 et S2 sont 2 droites alors B(S1,S2) est la bissectrice des 2 droites au sens où on l'entend habituellement.
- Si S1 est une droite et si S2 est un point alors B(S1,S2) est la parabole de directrice S1 et de foyer S2.

Algorithme de calcul du diagramme de Voronoï de n points
De nombreux algorithmes ont été proposés.dans la littérature.([P & S 85], [FOR 87]). Nous allons en détailler un. Il est dû à Preparata et est basé sur le principe de diviser pour conquérir. Son principe est le suivant :

- Partitionner S en 2 sous-ensembles S1 et S2 de tailles égales. On supposera que tous les points de S1 sont à "gauche" de ceux de S2.
- Construire récursivement les diagrammes Vor(S1) et Vor(S2).
- Construire la bissectrice B(S1,S2) de S1 et S2.
- Supprimer tous les arcs de Vor(S1) situés à droite de B(S1,S2) et tous les arcs de Vor(S2) à gauche de B(S1,S2).

La difficulté essentielle de cet algorithme est le calcul de la bissectrice de S1 et S2. Donnons tout d'abord une définition :

Définition Soient Conv(S1) et Conv(S2) les enveloppes convexes de S1 et S2. Soient D_{sup} et D_{inf} les 2 droites d'appui des 2 enveloppes convexes et soient t_1,t_2 (resp. r_1,r_2) les sommets en contacts avec D_{sup} (resp. D_{inf}).

La bissectrice de S1 et S2 est une chaîne polygonale et est calculée comme suit :

(1) • La médiatrice courante est initialisée avec la médiatrice de t_1 et t_2

(2) • On calcule la plus proche intersection I entre la médiatrice courante et les arcs de Vor(S1) ou Vor(S2).

(3) • Ajouter à B(S1,S2) la portion de médiatrice jusqu'à I.

(4) • Si I appartient à un arc de Vor(S1) alors la médiatrice courante devient la médiatrice entre le point de S1 adjacent par rapport au segment intersecté et le même point de S2. Même raisonnement si I appartient à un arc de Vor(S2).

(5) • Recommencer au pas 2 tant que la médiatrice courante n'est pas celle de r_1 et r_2.

La figure 2 donne un exemple de construction de B(S1,S2).

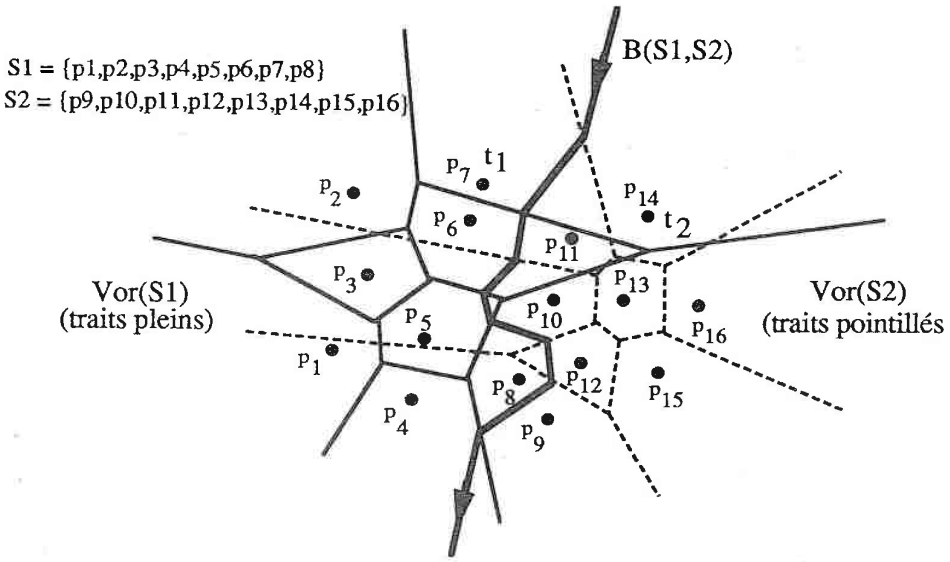


Figure 2 : Construction de la bissectrice de S1 et S2

Complexité

L'astuce principale de cet algorithme réside dans le fait que les arcs de Vor(S1) et de Vor(S2) seront au plus examinés une fois lors des recherches des intersections entre la médiatrice courante et les arcs. Donc la construction de B(S1,S2) coûte O(n). La récursivité engendrant un log, la complexité est en O(n.log(n)).

Applications

Le Diagramme de Voronoï est un outil remarquablement puissant pour résoudre des problèmes de proximité. Dans le plan, il représente une subdivision plane qui après un prétraitement en O(n.log(n)) permet de localiser un point en log(n). Les problèmes suivants peuvent alors se résoudrent :

• Trouver pour chaque point son plus proche voisin. Il suffit, une fois le diagramme construit, d'examiner pour un certain point uniquement les points qui définissent des arcs avec ce point. Chaque arc étant examiné au plus 2 fois, l'algorithme est linéaire et globalement le problème se résoud en O(n.log(n)).

• Il suffit de calculer un minimum tout en résolvant le problème précédent pour trouver le couple de points le plus proche. Une complexité en log(n.log(n)) apparaît encore ici. Utilité en contrôle aérien (les 2 avions les plus proches risquent une collision).

• Le plus petit arbre reliant les n points est facilement extrait du dual du diagramme de Voronoï. Le dual, appelé triangulation de Delaunay, est une triangulation dont les sommets sont les points eux mêmes et est telle que 2 sommets sont reliés si et seulement si leur cellule de Voronoï sont adjacentes (voir figure 3). Le plus petit arbre est recherché sur cette triangulation à l'aide de l'algorithme de Kruskal [KRU 56]. Il est trouvé en O(n.log(n)).

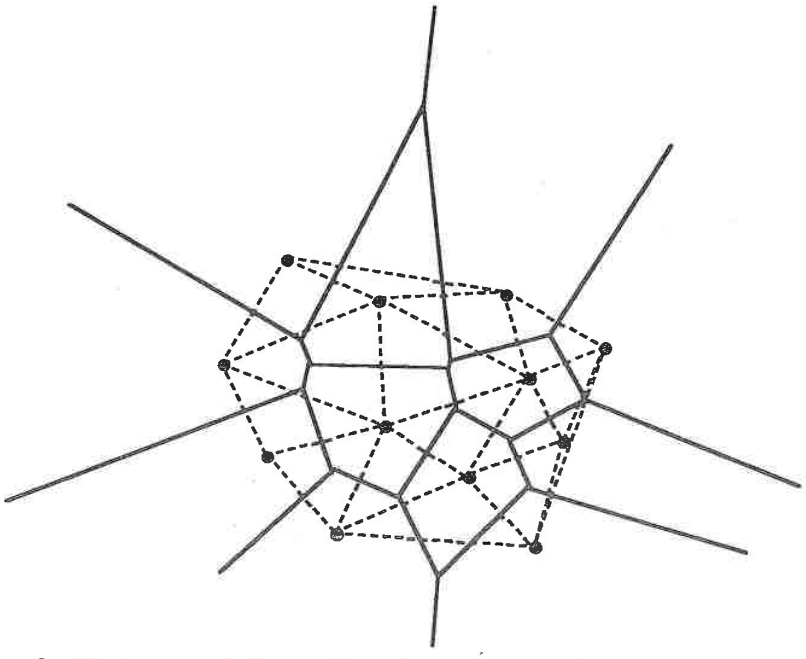


Figure 3 : Diagramme de Voronoï et triangulation de Delaunay (en pointillés)

• Une triangulation de n points, telle que tout cercle circonscrit à un triangle quelconque est vide, peut-être trouvée en O(n.log(n)). Il s'agit de la triangulation de Delaunay.

Conclusion Le diagramme de Voronoï de points est un outil puissant dans bon nombre de problèmes. De nombreux algorithmes existent à ce jour. La meilleure complexité de l'algorithme le calculant est en O(n.log(n)).

3.2 Calcul dynamique du diagramme de Voronoï d'un ensemble de segments

Nous présentons ici un algorithme dynamique pour le calcul du diagramme de Voronoï d'un ensemble de segments dans le plan. Ce travail a été réalisé en commun avec J.D. Boissonnat et R.Schott (voir [ABS 90]). Le problème se pose de la façon suivante : étant donné le diagramme de Voronoï de n-1 segments, quelles sont les modifications à apporter pour obtenir le diagramme après ajoût d'un nième segment ? Des algorithmes classiques existent déja (voir [L & D 81]). Des difficultés supplémentaires surgissent. D'une part, il est impossible d'ordonner les segments donc de les séparer en 2 ensembles disjoints. D'autre part, la bissectrice de 2 ensembles de segments peut être composée de plusieurs morceaux. La difficulté est de n'en oublier aucun ! La meilleure complexité est en $O(n.\log^2(n))$. Peu d'efforts ont concerné des algorithmes dynamiques alors que l'intérêt est évident. Les diagrammes de Voronoï sont très utilisés en planification de trajectoires. L'environnement d'un robot évoluant de manière dynamique, il apparait d'un grand intérêt de posséder des algorithmes dynamiques évitant le plus possible de calculs.

3.2.1 Définitions

Définition Un segment [a,b] est composé des 2 extrémités a et b et de la portion de droite notée]a,b[située entre ces 2 extrémités.

Définition La distance du point q au segment [a,b] est égale au min(d(q,q')), $q' \in [a,b]$ et d(q,q') est la distance euclidienne classique entre les 2 points q et q'. Si la projection q_p de q sur la droite (a,b) appartient à [a,b] alors la distance vaut $d(q,q_p)$ sinon la distance vaut le min de d(q,a) et de d(q,b).

Définition Le segment [a,b] définit les 3 régions V(a), V([a,b]) et V(b) qui sont telles que :
- $\forall q \in V(a)$, d(q,[a,b]) = d(q,a)
- $\forall q \in V([a,b])$, $d(q,[a,b]) = d(q,q_p)$ où q_p est la projection de q sur (a,b).
- $\forall q \in V(b)$, d(q,[a,b]) = d(q,b)

La figure 4 illustre ces définitions.

Définition La bissectrice des 2 segments [a,b] et [c,d] est l'ensemble des points équidistants de [a,b] et [c,d]. Elle peut être constituée de 3 types de morceaux :
- Dans une région de type $V(x) \cap V(y)$ (x et y sont des extrémités des 2 segments), la portion de bissectrice est la médiatrice de x et y.

• Dans la région V([a,b]) ∩ V([c,d]) la portion de bissectrice est la bissectrice des 2 droites (a,b) et (c,d).

• Dans une région de type V(x) ∩ V([z,t]) la portion de bissectrice est la parabole de foyer x et de directrice la droite (a,b).

Voir la figure 5.

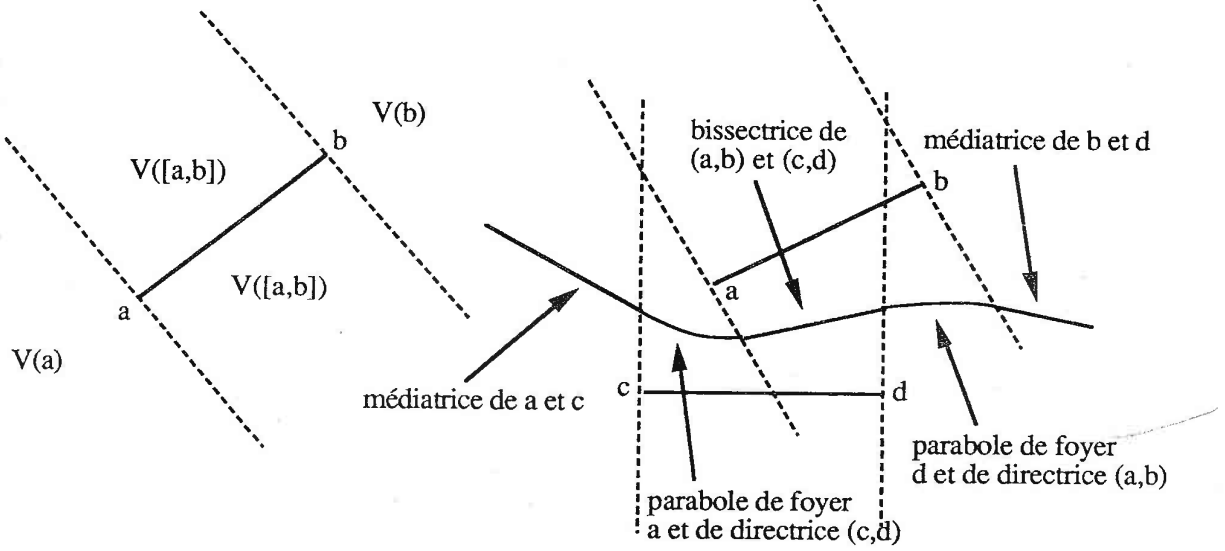


Figure 4 : Les régions engendrées par un segment

figure 5 : bissectrice de 2 segments

Définition Soit $h(s_i,s_j)$ le demi-plan constitué des points situés plus près du segment s_i que du segment s_j. Le demi-plan est à prendre au sens large. Il est délimité par la bissectrice de s_i et s_j. Le diagramme de Voronoï de l'ensemble S de segments $\{s_1,...,s_n\}$ est égal à l'union des $V(s_i)$ où $V(s_i) = \cap\ h(s_i,s_j)$, $i \neq j$.

Définition L'enveloppe convexe de S est égale à l'enveloppe convexe des points extrémités des segments de S.

Nous allons énoncer ici 2 résultats utiles à l'algorithme qui suivra :

Résultat Lee et Drysdale dans [L & D 81] annoncent que le nombre d'arcs et de sommets du diagramme de Voronoï d'un ensemble S d'objets quelconques est en O(n).

Résultat Soit V(si) une cellule du diagramme de Voronoï de S. Soit $\{V(t_1),V(t_2),...,V(t_k)\}$ (les t_i sont des renomages des s_i) l'ensemble des cellules adjacentes à $V(s_i)$ et telles que :

- $V(t_1)$ adjacente à $V(t_2)$
- $V(t_2)$ adjacente à $V(t_3)$
- ...
- $V(t_k)$ adjacente à $V(t_1)$

Les cellules $V(t_j)$ sont ordonnées sur la frontière de $V(s_i)$ et engendrent la suite ordonnée CW $r_1, r_2, ..., r_k$ de sommets de Voronoï sur la frontière de $V(s_i)$. Il existe une façon de parcourir les r_j en utilisant au plus une fois chaque arc du diagramme et ceci sans utiliser les arcs de $V(s_i)$. Soit r_1 le sommet adjacent aux cellules $V(s_i)$, $V(t_1)$ et $V(t_k)$. Le sommet r_2 sera atteint en parcourant CW les arcs de la cellule $V(t_1)$. r_3 sera atteint en parcourant CW les arcs de la cellule $V(t_2)$ etc... Un tel parcours n'utilise jamais 2 fois un même arc vu l'ordonnancement des cellules $V(t_j)$ sur la frontière de $V(s_i)$. La figure 6 donne le parcours numéroté des arcs des $V(t_j)$ ainsi que la suite des r_j.

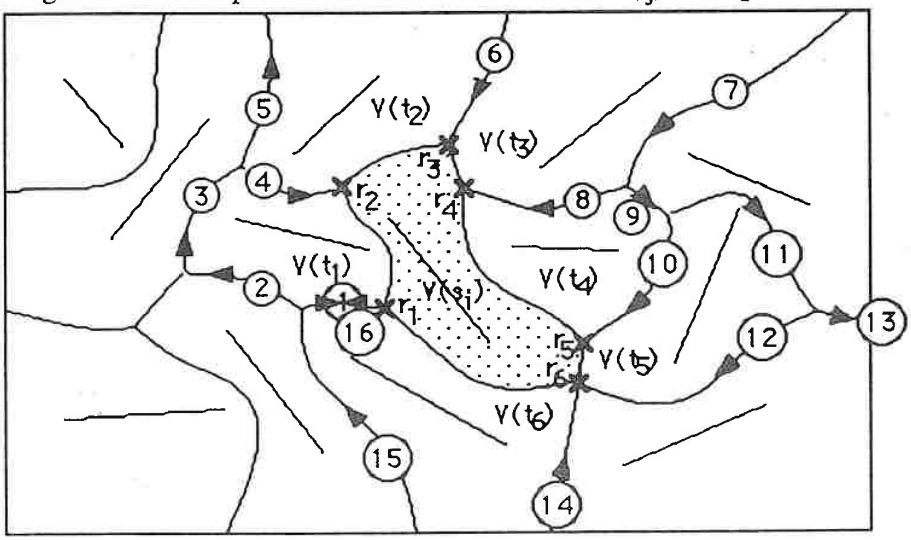


Figure 6 : Parcours des rj sur la frontière de $V(s_i)$

Remarque Si une branche infinie est rencontrée sur la frontière d'un V(tj) alors l'arc suivant est simplement l'autre branche infinie.

Résultat Soit [a,b] le nouveau segment introduit.et soit $V(s_c)$ une cellule du diagramme existant intersectée par [a,b]. Le nombre d'intersections entre la bissectrice de s_c et [a,b] et la frontière de V(sc) est au plus égal à 2.

Preuve Voir la figure 7. Par l'absurde. soient A,B,C les 3 intersections entre la bissectrice de s_c et [a,b] avec $V(s_c)$. Soit s_2 le segment adjacent à S_c par rapport à l'arc intersecté.
- $d(B,s_c) = d(B,[a,b])$ et $d(B,s_2) = d(B,s_c)$ donc $d(B,s_2) = d(B,[a,b])$ donc le point B appartient à la bissectrice de s_2 et [a,b]. Par le même raisonnement, on peut dire que le point C appartient à la

bissectrice de s_2 et [a,b]. Cette bissectrice passe donc par C, B et un 3ième point D. Pour des raisons de continuités, elle doit avoir l'allure de celle présentée à la figure 7.

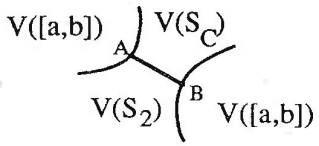
• Les points y sont situés plus près de s_2 que de sc et plus prés de s_2 que de [a,b] donc ils appartiennent à $V(s_2)$.

• En employant le même argument, les points z appartiennent à $V(s_c)$.

• Les points t sont situés plus près de [a,b] que de sc et plus prés de [a,b] que de s_2 donc ils appartiennent à V([a,b]).

• En employant le même argument, les points x appartiennent à V([a,b]).

On peut résumer la situation par le schéma suivant :



On a V([a,b]) scindé en 2 ce qui est contraire à la propriété de connexité des cellules de Voronoï.

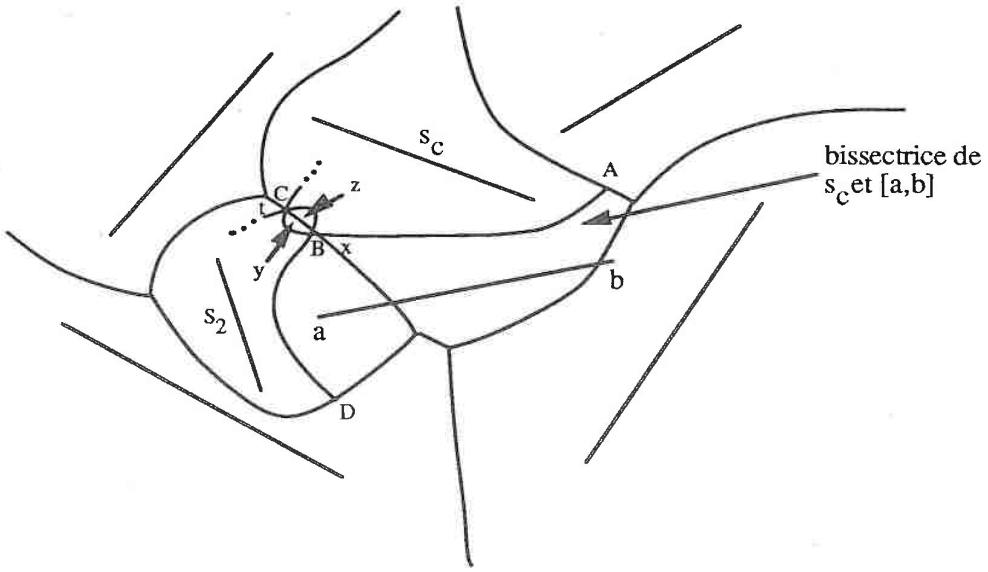


Figure 7 : La bissectrice courante intersecte au plus 2 fois une cellule existante

Remarque Pour uniformiser les traitements, nous avons inclus le diagramme dans un cercle. De cette façon, on peut considérer qu'il n'y a plus de branches infinies et que les recherches d'intersections entre la bissectrice courante et les arcs du diagramme aboutiront toujours. Il y a donc, dans le cas de 2 branches infinies, un arc fictif les reliant et qui vaut la portion de cercle entre les 2 intersections du cercle avec les branches infinies (Voir figure 9).

3.2.2 Algorithme de calcul du diagramme

Déroulement intuitif à la main (voir figure 8).

Soit [a,b] le nouveau segment introduit (la nouvelle cellule sera constituée CW). Soit $V(s_i)$ la cellule contenant l'extrémité a. La bissectrice de s_i et [a,b] intersecte $V(s_i)$ en $I_{si,sj}$ et $I_{st,si}$ (supposons que $I_{st,si}$ soit CW par rapport à $I_{si,sj}$). La cellule $V(s_i)$ est mise à jour à l'aide de la bissectrice. La nouvelle cellule à examiner devient $V(s_t)$. La bissectrice entre s_t et [a,b] intersecte $V(s_t)$ en $I_{st,si}$ et $I_{st,sk}$. La cellule $V(s_t)$ est mise à jour à l'aide de cette bissectrice. On répète ce processus tant que la nouvelle intersection n'est pas égale à $I_{si,sj}$ qui est la première intersection trouvée.

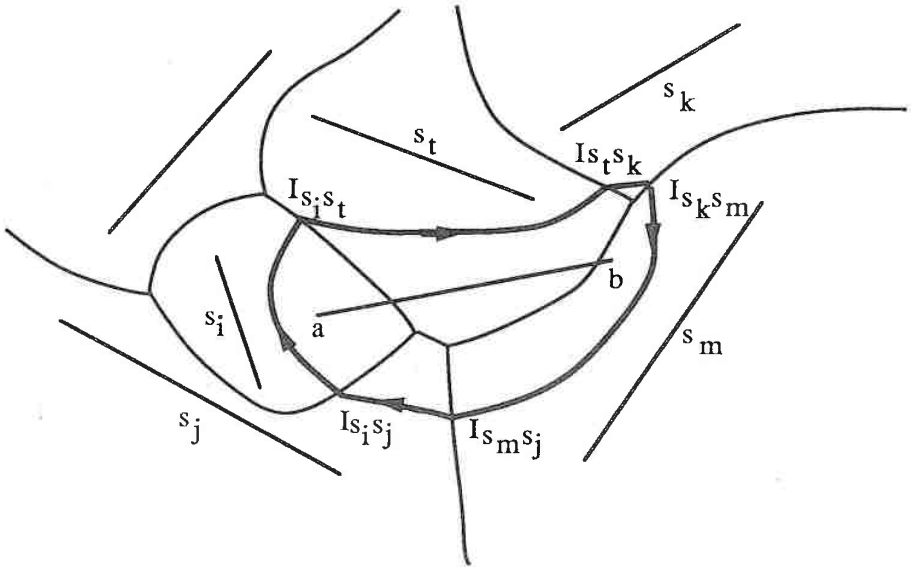


Figure 8 : Déroulement de l'algorithme

Remarque 2 cas se présentent :

- [a,b] appartient à l'enveloppe convexe des n-1 segments de la scène. V([a,b] ne posséde pas d'arcs infinis. L'algorithme s'arrêtera lorsque la première intersection sera atteinte.
- [a,b] n'appartient pas à l'enveloppe convexe des n-1 segments donc V([a,b]) possède 2 branches infinies et l'algorithme s'arrêtera lorsqu'elles auront toutes 2 été trouvées sur le cercle englobant (voir figure 9).

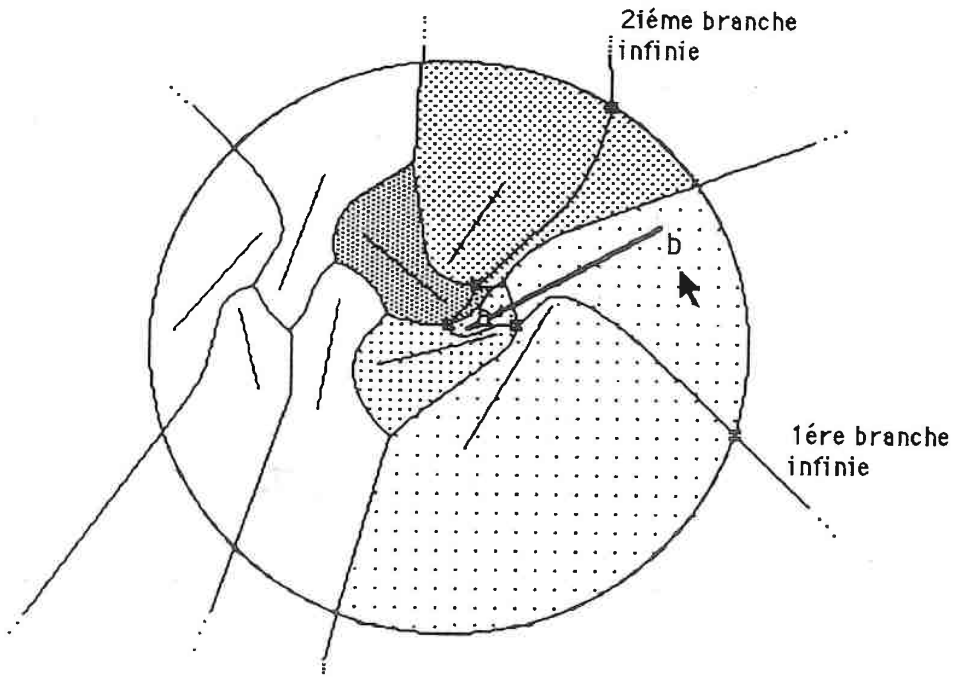


Figure 9 : Gestion des branches infinies

Remarque Une cellule de Voronoï est constituée d'un ensemble d'arcs où un arc est un morceau d'une certaine bissectrice entre 2 segments. Le morceau peut être constitué de plusieurs types de courbes (segments ou paraboles). On supposera que la connaissance du diagramme est définie par la connaissance de toutes les cellules et que la connaissance d'une cellule $V(s_i)$ est fournie par les 2 fonctions Suivant et Précédent qui chacune, pour un arc donné de $V(s_i)$, fournissent respectivement l'arc suivant et précédent sur $V(s_i)$. On supposera que Suivant définit un sens CCW sur la cellule et que Précédent définit un sens CW.

Algorithme

En entrée : $V(S)$ où $S = \{s_1,...,s_{n-1}\}$
[a,b] le nouveau segment introduit
En sortie : $V(S \cup [a,b])$

INITIALISATIONS

(0) c <- Cellule de Voronoï contenant a.
Nb_Branches_Infinies = 0 *(* comptabilisation du nombre de branches infinies trouvées *)*
I_{Deb},I_{Fin} <- les 2 intersections entre la bissectrice de s_C et [a,b] avec $V(s_C)$.
Si I_{Deb} et I_{Fin} définissent un sens CW sur $V(s_C)$ alors les permuter.
Premier_I <- I_{Deb}

Arret <- faux

Tant que non arret Repeter

(1) *Arret <- (I_{Fin} == Premier_I) ou (I_{Fin} appartient au cercle)*

(2) *Mettre à jour $V(s_C)$ à l'aide de la portion de bissectrice comprise entre I_{Deb} et I_{Fin} (notée $B(I_{Deb},I_{Fin})$), soit :*

Soit VI_{Deb} l'arc de $V(s_C)$ contenant I_{Deb}

Soit VI_{Fin} l'arc de $V(s_C)$ contenant I_{Fin}

Suivant(VI_{Deb}) <- $B(I_{Deb},I_{Fin})$

Précédent($B(I_{Deb},I_{Fin})$) <- VI_{Deb}

Suivant($B(I_{Deb},I_{Fin})$) <- VI_{Fin}

Précédent(VI_{Fin}) <- $B(I_{Deb},I_{Fin})$

(3) *Mettre à jour V([a,b]) à l'aide de $B(I_{Deb},I_{Fin})$ soit:*

Précédent(B_{Pred}) <- $B(I_{Deb},I_{Fin})$ si B_{Pred} est définie.

Suivant($B(I_{Deb},I_{Fin})$) <- B_{Pred} si B_{Pred} est définie.

(4) *B_{Pred} <- $B(I_{Deb},I_{Fin})$*

si I_{Fin} n'est pas sur le cercle

alors *c <- Cellule adjacente à c à l'intersection I_{Fin}*

(6) *I_{Deb} <- I_{Fin}*

(7) *I_{Fin} <- autre intersection entre la bissectrice de s_C et [a,b] avec $V(s_C)$*

(8) *Si I_{Deb} et I_{Fin} définissent un sens CW sur $V(s_C)$ alors les permuter.*

Fin Tant que

(9) si I_{Fin} n'est pas sur le cercle

alors *(* terminaison du circuit *)*

Suivant(Premiere Bissectrice <- derniere bissectrice

Précédent(dernière Bissectrice <- Premiere bissectrice

sinon *si I_{Deb} n'est pas sur le cercle*

alors *(* recommencer un processus analogue en partant du point Premier_I jusqu'à recontre d'une seconde branche infinie. Le sens de parcours étant l'inverse, les appels aux fonctions Suivant et Precedent devront être permutés *)*

Dernier_I <- I_{Fin}

Dernnière_Bissectrice <- $B(I_{Deb},I_{Fin})$

Tant que . . .

Fin Tant que

(raccordement des 2 branches infinies par la portion de cercle entre les points Dernier_I et I_{Fin} *)*
Précédent(Derniere_Bissectrice) <- portion du cercle entre Dernier_I et I_{Fin}
Suivant(portion du cercle) <- Derniere_Bissectrice
Précédent(portion du cercle) <- $B(I_{Deb},I_{Fin})$
Suivant($B(I_{Deb},I_{Fin})$) <- portion du cercle

sinon *(* la bissectrice est constituée d'une seule portion. voir figure 13 *)*
Précédent($B(I_{Deb},I_{Fin})$) <- portion du cercle entre I_{Deb} et I_{Fin}
Suivant(portion du cercle) <- $B(I_{Deb},I_{Fin})$
Précédent(portion du cercle) <- $B(I_{Deb},I_{Fin})$
Suivant($B(I_{Deb},I_{Fin})$) <- portion du cercle

(10) Mise à jour de la subdivision
Fin

Complexité

Le pas (0) consiste à localiser l'extrémité a dans une subdivision plane. On sait que (voir [P & T 89]) l'intérrogation coûte $O(\log^2 n)$ et que le coût de la mise à jour de la subdivision après insertion d'un nouveau segment est en $O(\log^2 n)$ puisque le segment ajouté peut être considéré comme une chaîne de longueur 1.
L'itération principale est exécutée T fois, où T est le nombre de cellules adjacentes à V(a,b) dans le diagramme final.
A l'intérieur de la boucle, les pas 1,2,3,4,5,6,8 s'éxécutent en temps constant. T est en O(n) dans le pire cas.
Le pas 7 consiste à trouver l'autre intersection de la bissectrice dans la cellule courante. Globalement, la suite des intersections pourra être obtenue en examinant au plus une fois chaque arc du diagramme vu le résultat énoncé précédement. Partant de la première intersection la seconde sera trouvée en parcourant CW les arcs de la cellule intersectée. Donc globalement, les recherches des secondes intersections se feront en O(n).
Le pas (9) se fait en temps constant sauf dans le cas où l'on doit rechercher une seconde branche infinie. Dans ce cas, il s'agit d'une répétition de la boucle tant que précédente. Cela ne modifie pas la complexité globale.
Le coût du pas (10) est en $O(\log^2 n)$. ([P & T 89])
On a donc pour la création d'un nouveau diagramme, une complexité en $O(\log^2 n) + O(n) + O(\log^2 n)$ soit O(n).

Pour la création du diagramme de n segments on a une complexité en n*O(n) soit $O(n^2)$.
Qu'en est-il en moyenne ? La formule d'Euler dit que tout graphe planaire à n sommets a au plus 3n-6 arcs et au plus 2n-4 faces. La triangulation de Delaunay, duale du diagramme de Voronoï des segments est un graphe planaire donc elle satisfait à la formule d'Euler.
Chaque arc de Delaunay marquant un arc du diagramme de Voronoï, on en déduit immédiatement que le diagramme de Voronoï a au plus 3n-6 arcs. Un arc appartient à 2 faces donc on peut dire qu'on dispose d'au plus 2.(3n-6) arcs propres à une face soit 6n-12 arcs. Sachant que l'on a n faces au diagramme, une face aura en moyenne (6n-12)/n soit 6 - 12/n arcs ce qui donne 6 quand n tend vers l'infini et de toute façon un nombre moyen plus petit que 6 quelque soit n.
Donc en moyenne T est constant et par là le nombre des 2ièmes intersections aussi. On obtient donc une complexité en moyenne majorée par $O(n*\log^2 n)$.

Conclusion Nous avons présenté un algorithme dont la complexité en moyenne est majorée par $O(n*\log^2 n)$, c'est-à-dire qu'elle n'est pas plus mauvaise que celle de l'algorithme de type "divide and conquer" de Lee et Drysdale [L & D 81]. Cet algorithme doit pouvoir se généraliser aisément au cas d'un ensemble de polygones. Nous allons aborder maintenant le calcul de diagrammes discrets. Nous perdrons la notion d'exactitude mathématique mais nous gagnerons en simplicité, en souplesse et en puissance.

3.3 Calcul discret du diagramme de Voronoï d'un ensemble d'objets quelconques

3.3.1 Liens avec les algorithmes en traitement d'image

Nous allons présenter ici 2 algorithmes tirés de [PEP 90]. Nous montrons par la même qu'il peut exister des liens entre des domaines apparemment disjoints. Les 2 algorithmes étudiés ici sont des algorithmes de squelettisation. Il existe une similitude entre les notions de squelette et de diagramme de Voronoï. Du point de vue du traitement d'image, un squelette possède les définitions suivantes :

- Avec la technique dit du "feu de prairie", le squelette est l'ensemble des points atteints simultanément par au moins 2 fronts de feu lorqu'un feu est allumé simultanément en chacun des points de la frontière.
- Le squelette est l'ensemble des centres des boules maximales inclues dans l'environnement.
- Le squelette est l'ensemble des points équidistants d'au moins 2 points du bord de l'environnement et ces 2 points sont 2 plus proches points.
- Supposons qu'on associe en tout point (x,y) du plan la plus petite distance z du point à l'environnement. On obtient alors une fonction f(x,y,z) dont les crêtes constituent par projection sur le plan (xOy) le squelette recherché.

Ces quelques définitions sont toutes équivalentes entres elles et avec la définition du diagramme de Voronoï. Autre domaine, autres qualifications ....

• Premier algorithme

La dernière définition est celle à la base de cet algorithme. Soit P un point du plan discret et soient P1, P2, ...,P8 ses 8 voisins (voir figure 10).

P_4	P_3	P_2
P_5	P	P_1
P_6	P_7	P_8

Figure 10 : Voisins de P

Définissons d(P) comme la plus petite distance de P à l'environnement.

On décrètera que le point P appartient au squelette s'il vérifie la condition suivante :

d(P) > d(P1) et d(P) > d(P5) ou

d(P) > d(P2) et d(P) > d(P6) ou

d(P) > d(P3) et d(P) > d(P7) ou

d(P) > d(P4) et d(P) > d(P8)

On dira dans ce cas que P est un maximum directionnel.

Cette condition n'est pas correcte si certaines largeurs minimales entre des objets de l'environnement correspondent à un nombre pair de cellules discrètes. Dans ce cas, si le test précédent échoue, alors on effectuera ce test (voir figure 11) :

P et P3 ∈ Squelette si d(P) > d(P7), d(P) = d(P3) et d(P3) > d(P9)

P et P5 ∈ Squelette si d(P) > d(P1), d(P) = d(P5) et d(P5) > d(P10)

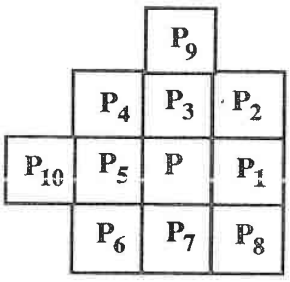


Figure 11 : Test pour les largeurs paires

Malgré tout, il se peut que ces 2 tests laissent des discontinuités dans le squelette. Pour combler ces lacunes le test d(P)>d(P5), d(P) = d(P1) et d(P) > d(P2) (que l'on dérive 8 fois par rotations

de 45°) peut être appliqué. Il sera mis en oeuvre à l'aide d'un automate qui aura pour souci de ne pas générer de branches parasites.
Cette méthode permet d'obtenir un squelette qui équivaut à un diagramme de Voronoï. Elle est simple mais sa mise au point est délicate. Une dernière étape transforme le squelette en un graphe de cellules exploitable par un algorithme.

• Second algorithme

Cette méthode exploite directement les segments présents dans la scène et applique sur chacun d'eux la technique du feu de prairie. Un segment pouvant être vu comme 2 extrèmitès et une portion de droite les reliant, il suffira de réaliser la propagation d'un onde linéaire pour les points non extrémités et circulaire pour les points extrémités (voir figure 12).

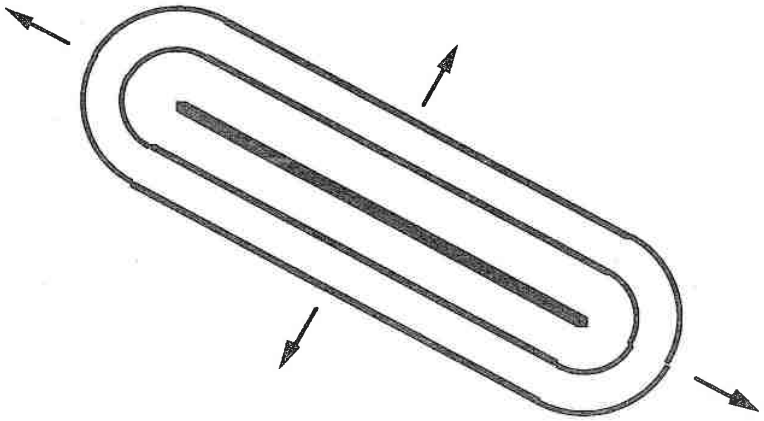


Figure 12 : Propagation d'une onde à partir d'un segment

Le principe du feu de prairie dit que les points où 2 ondes se rencontrent sont équidistants des 2 segments générateurs de l'onde. L'algorithme réalisant cette propagation est un algorithme qui travaille sur un réseau de processeurs très simples attachés chacun à une cellule discrète. L'état de chaque cellule est un couple (t,x), où t représente le nombre d'itérations et x la distance minimale. Les valeurs initiales sont (0,0) pour les cellules appartenant au bord de l'environnement et (0,∞) pour les autres. Chaque cellule envoie un jeton dans une direction donnée lorsque t et x vérifient une condition précise. Une cellule reçoit un jeton si x = ∞.
On a :

- Si x = t-2, un jeton est émis horizontalement et verticalement. La cellule reçevant le jeton prend l'état (t+1,t+1) à la cadence suivante.
- Si x = t-3, un jeton est émis dans les 4 directions diagonales. La cellule recevant le jeton prend l'état (t+1,t+1) à la cadence suivante.
- Dans les autres cas, la variable t est incrémentée.

Dans la réalité, l'état de chaque cellule sera augmenté d'une variable α désignant un certain segment.afin de ne pas retenir les croisements d'onde relatifs à un même segment.
Nous ne parlerons pas plus des algorithmes issus du traitement d'image. Nous allons proposer un algorithme basé sur les mêmes principes et sensiblement plus simple.

Il apparaît, après les études précédentes, que plus les objets deviennent complexes, plus les algorithmes le deviennent aussi. Nous allons proposer un algorithme très simple insensible à la nature des objets dont il cherche à calculer le diagramme. Il sera doté d'une souplesse permettant d'obtenir des diagrammes légèrement différents suivant l'utilité demandée (existence ou non d'arcs dans les parties concaves des polygones par exemple). Il sera capable de calculer des diagrammes d'objets hétérogènes. Il sera dynamique et sa robustesse largement supérieure à celle des algorithmes connus (travail sur des entiers). Ses inconvénients seront d'une part la perte de la notion d'exactitude (vu l'univers discret) - celle-ci pouvant malgré tout être levée partiellement - et d'autre part des performances moindres que celles des algorithmes classiques. Nous proposerons un algorithme parallèle qui a été implanté sur une machine T_Node pour montrer qu'ils peuvent néanmoins être rapides. Enfin, leur simplicité devrait être compatible avec un câblage (architectures spécialisées).
Nous allons définir briévement ici l'univers discret dans lequel nous évoluons. Une généralisation sera proposée au début du chapitre 4.

3.3.2 Définitions

Définition L'univers de travail sera un pavage de $\mathbb{R}^2$ par des cellules carrées de dimensions h. Une cellule sera définie par un couple (i,j) d'entiers et vaudra l'ensemble des (x,y) de $\mathbb{R}^2$ tels que $i*h \leq x \leq (i+1)*h$ et $j*h \leq y \leq (j+1)*h$. Elle sera notée Cel(i,j). Elle possède 4 voisins (haut, bas, gauche, droit) lorsque l'on est dans la relation de voisinange dite de 4-connexité et en possède 8 (ajoût aux précédents des 4 cellules diagonales) lorsque l'on est en 8-connexité. Nous nous placerons pour la suite en 8-connexité.

Définition Soit S un ensemble d'objets dans le plan. Soit V(S) le diagramme de Voronoï de S. On appellera diagramme discret de V(S) l'ensemble des cellules du plan intersectées par V(S) (voir figure 13 à gauche).

Le but est de construire le diagramme discret sans avoir à construire le diagramme continu.

Définition Soit $\{(a_1,b_1),(a_2,b_2),...,(a_n,b_n)\}$ une suite de cellules telles que (a_i,b_i) voisine de (a_{i+1},b_{i+1}) pour tout i dans [1,n-1]. On dira que cette suite est d'épaisseur 1 si et seulement si toute cellule a au plus 2 cellules voisines.

Définition Soit (i,j) une cellule du plan discret et soit S un ensemble d'objets On appellera influence de (i,j) (notée influence(i,j)) le plus proche objet de (i,j) (voir figure 13 à droite). La distance de la cellule (i,j) à un point quelconque du plan réel sera égale à la distance euclidienne entre le centre (i*h + h/2 , j*h + h/2) et le point considéré.
Les objets, dans le cas d'un ensemble de polygones, peuvent signifier les polygones eux mêmes, les segments ou bien encore des groupements spéciaux de segments.

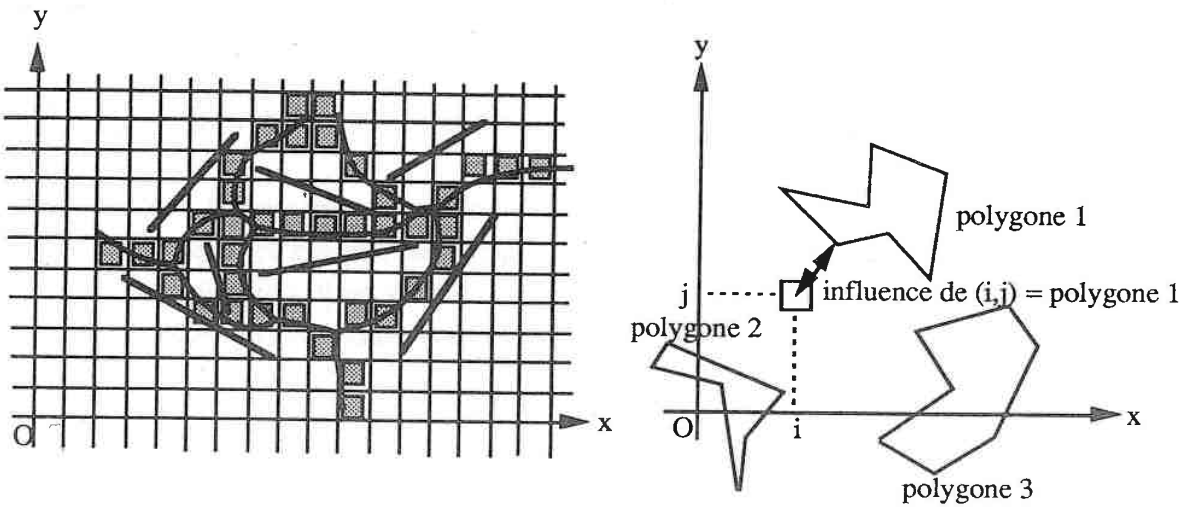


Figure 13 : Diagramme de Voronoï discret et influence d'une cellule lorsque les objets considérés sont des polygones.

Calcul des influences

Soit S un ensemble d'objets et soit (x,y) un point quelconque. Le calcul de l'influence du point (x,y) revient au calcul d'une distance minimum entre (x,y) et les objets de S. Il suffit de savoir calculer la distance du point (x,y) à un objet s quelconque.

- Si s est un point alors la distance est la distance euclidienne entre (x,y) et s.
- Si s est un segment alors se rapporter à la définition donnée en § 3.2.1.
- Si s est un polygone alors la distance est égale au min des distances de (x,y) à chacun des segments du polygone.

• Si s est un cercle alors la distance est égale à la distance entre (x,y) et le centre du cercle moins le rayon.

Le type d'objet induit une certaine fonction de calcul de la distance. Celle-ci devient un paramètre général du problème.

Définition Le critère d'appartenance d'une cellule du plan discret au diagramme de Voronoï est le suivant : la cellule (i,j) appartiendra au diagramme si et seulement si elle se trouve à la frontière de 2 cellules de Voronoï. Une frontière est une ligne telle qu'à son niveau on passe d'une cellule à une autre. Ce changement "d'état" se mesurera à l'aide des influences de la cellule elle même et celles de 2 de ses voisins. Celui de gauche et celui du haut. Autrement dit, si Influence(i,j) ≠ Influence(i-1,j) ou si Influence(i,j) ≠ Influence(i,j+1) alors on décrètera que Cel(i,j) appartient au diagramme de Voronoï (voir figure 14).

3.3.3 Diagrammes personnalisés

Supposons que l'environnement soit composé d'un ensemble de polygones quelconques. La définition stricte du diagramme des polygones donnera un diagramme sans arc dans les parties concaves des polygones. Il pourrait être intéressant qu'il y en ait. La figure 15 donne quelques exemples de plusieurs diagrammes pour un même environnement.

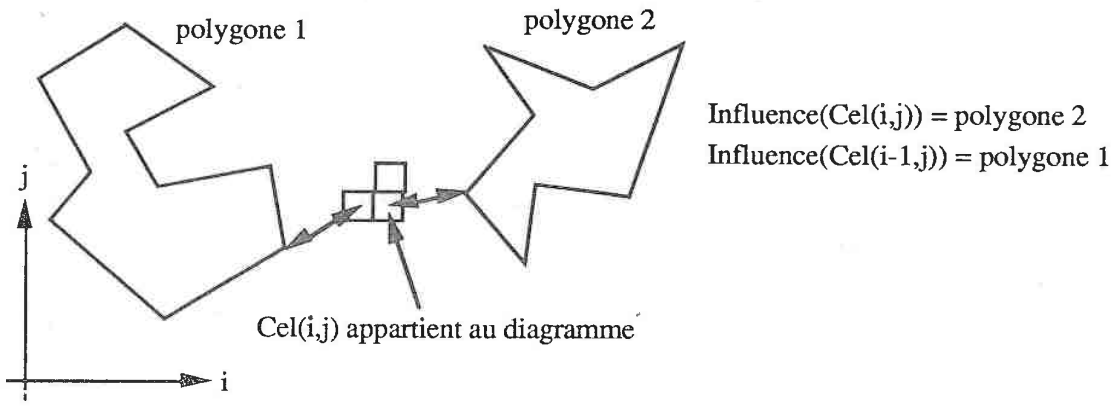


Figure 14 : Changement d'influence

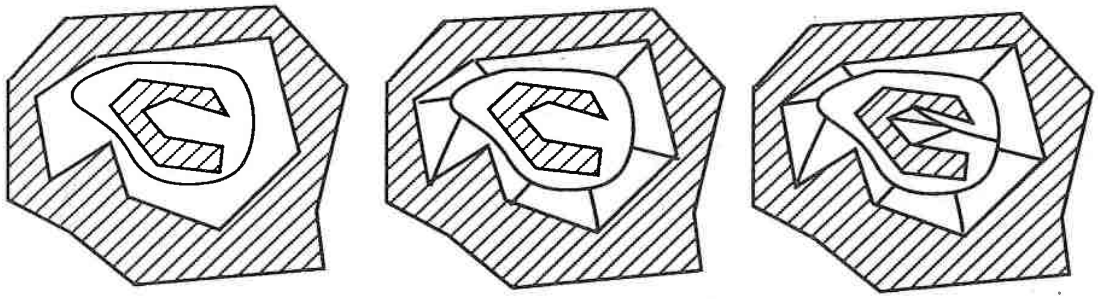


Figure 15 : Diagrammes de Voronoï personnalisés

Cette personnalisation peut être obtenue de 2 façons :

• En introduisant une numérotation des segments : si le changement d'influence est dû à 2 segments de numéros différents alors la cellule sera retenue sinon elle ne sera pas retenue. La figure 16 donne 2 exemples d'une telle numérotation et les diagrammes correspondants.

• A l'aide de critères géométriques : par exemple, si le changement d'influence est dû à 2 segments formant une concavité ou s'ils appartiennent à 2 polygones différents alors la cellule sera retenue sinon non.

Définition Supposons que tous les sommets des polygones soient fournis dans l'ordre CW. Les 3 sommets consécutifs V1,V2,V3 du polygone englobant forment une concavité si V3 est à droite de la droite orientée (V1,V2). Dans le cas contraire, V1,V2,V3 forment une convexité. Les 3 sommets consécutfs V1,V2,V3 d'un polygone quelconque englobé forment une concavité si V3 est à gauche de la droite orientée (V1,V2). Dans le cas contraire, V1,V2,V3 forment une convexité.

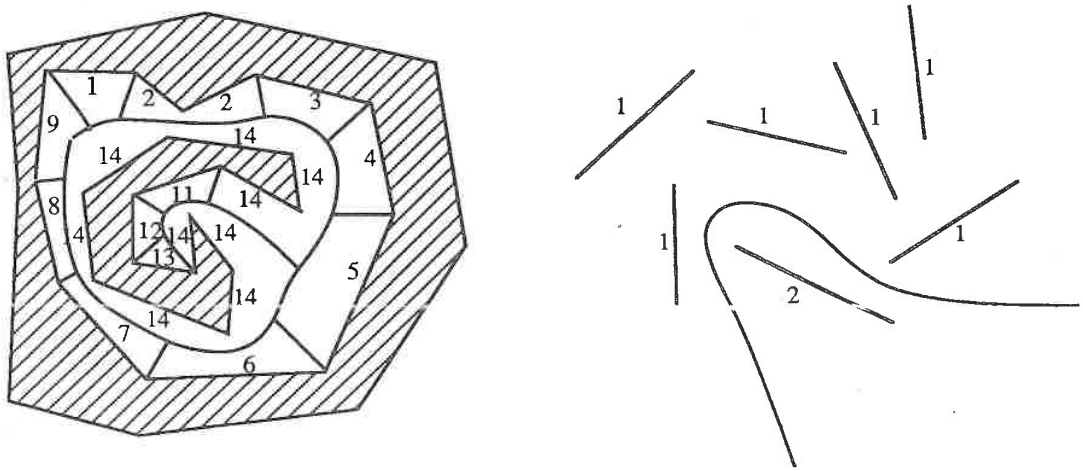


Figure 16 : Numérotation et diagrammes correspondant

3.3.4 Algorithme

Idée : les arcs du diagramme sont obtenus par un suivi récursif. Soit (i,j) la dernière cellule otenue sur un certain arc du diagramme. La ou les cellule(s) suivante(s) de l'arc est (sont) parmi les 8 voisins. Il suffit de les examiner tous et de détecter pour chacun un changement d'influence. Rappelons que le changement d'influence se détecte au niveau d'une cellule par l'examen des influences de ses voisines de gauche et du haut. Les nouvelles cellules trouvées sont empilées puis examinées récursivement. En particulier, un sommet du diagramme de Voronoï est tel qu'il existe plusieurs cellules voisines (voir figure 17)

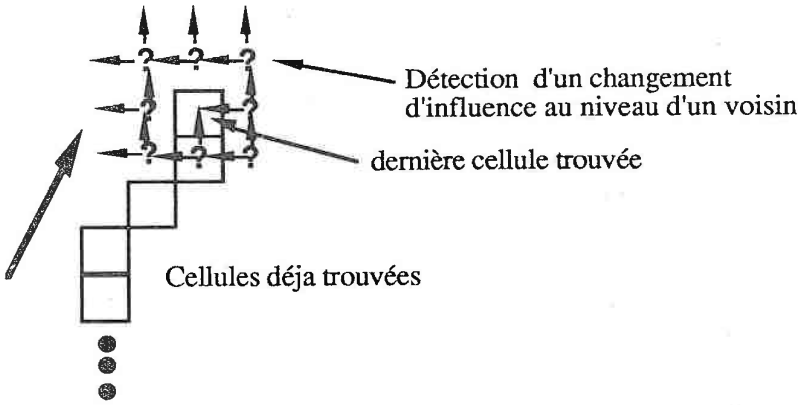


Figure 17 : Principe de base de l'algorithme par suivi récursif

L'initialisation consiste à obtenir une première cellule. Peu importe la méthode. Cette première cellule sera le germe de l'algorithme.

L'algorithme devient alors le suivant :

Fonction de service : Critère(objet1,objet2) rend vrai s'il doit exister un diagramme entre objet1 et objet2. Cette fonction personnalise le diagramme et est définie par l'utilisateur. Rappelons qu'elle se base sur une numérotation des objets ou sur des considérations géométriques.

En entrée : Un ensemble S d'objets quelconques.
En sortie : Le diagramme de Voronoï discret se présentant comme une suite connexes, d'épaisseur 1, de cellules du plan discret.

Début
Pile ← *Pile_Vide*
Diagramme ← *Diagramme_Vide*

```
Choix d'un point (Ideb,Jdeb) appartenant au diagramme
Empiler(Pile,(Ideb,Jdeb))
Ajouter(Diagramme,(Ideb,Jdeb))

Tant que Pile non vide répéter
        Sommet ← Dépiler(Pile)
        Pour (Px,Py) dans les voisins de Sommet répéter
                Si (Px,Py) n'appartient pas déja au diagramme
                        alors   Ifl_G = Influence(Px-1,Py)  /* influence voisin de gauche de Px,Py */
                                Ifl = Influence(Px,Py)      /* influence de Px,Py */
                                Ifl_H = Influence(Px,Py-1)  /* influence voisin du haut de Px,Py */
                                si (Ifl_G ≠ Ifl  et Critère(Ifl_G,Ifl)) ou (Ifl_H ≠ Ifl  et Critère(Ifl_H,Ifl))
                                        alors   Ajouter(Diagramme,(Px,Py))
                                                Empiler(Pile,(Px,Py))
                                fsi
                fsi
        fpour
ftantque
```

Remarques

• La connexité du diagramme de Voronoï implique que l'algorithme ci-dessus déterminera tous les arcs du diagramme sans en oublier.

• Le diagramme déterminé par cet algorithme se présente sous la forme d'une suite de cellules connexes auxquelles on peut ajouter de l'information : il est possible de labeller chaque cellule par le couple d'objets donnant naissance à la cellule. En effet, soient [a,b] et [c,d] les 2 segments provoquant le changement d'influence au niveau de la cellule (i,j). Si (i,j) ∈ V(a,b) et (i,j) ∈ V(c,d) alors on sait qu'on est sur la bissectrice des 2 droites (a,b) et (c,d) et donc sur un segment. Si (i,j) ∈ V(a) et (i,j) ∈ V(c,d) alors on sait qu'on est sur la parabole de foyer a et de directrice (c,d) etc. Il est possible dés lors, de modèliser le diagramme sous la forme d'un graphe d'éléments qui sont soit des segments de droite, soit des arcs de parabole. Ceci valant bien sur pour le diagramme d'un ensemble de segments ou de polygones.

• Les arcs produits par le diagramme ne sont pas d'épaisseur 1. Il convient de modifier légèrement la règle qui décide si une cellule doit ou non faire partie du diagramme. Par souci de clarté, nous ne l'avons pas fait figurer dans l'algorithme.

Complexité et conclusion

Soit n le nombre d'objets de la scène. Soit K le nombre total de cellules de Voronoï. Pour chacune des cellules un calcul de minimum sur les n objets est nécessaire. La complexité est donc en O(K.n).

Nous avons présenté un algorithme particulièrement simple pour le calcul du diagramme de Voronoï discret d'objets quelconques. Il est néanmoins possible d'obtenir un diagramme sous la forme d'un graphe de primitives de base. Sa simplicité (la seule fonction est celle permettant de calculer la distance d'un point à un objet de la scène) est gage de robustesse. Il est insensible aux cas dégénérés qui grèvent tant les algorithmes classiques. La souplesse fournie par la numérotation permet aisément de le rendre dynamique. En effet, soit le diagramme de n-1 objets. En attribuant aux n-1 objets un même numéro et au nième ajouté un autre numéro, l'algorithme calcule la bissectrice entre les n-1 objets et le nouvel objet. La bissectrice étant construite pas à pas, les intersections entre la bissectrice et le diagrame existant sont facilement calculables. Les mises à jour du diagramme sont alors possibles.

Le chapitre 4 donnera un exemple d'utilisation du diagramme de Voronoï. Les figures 18 et 19 donnent des exemples de diagrammes discrets.

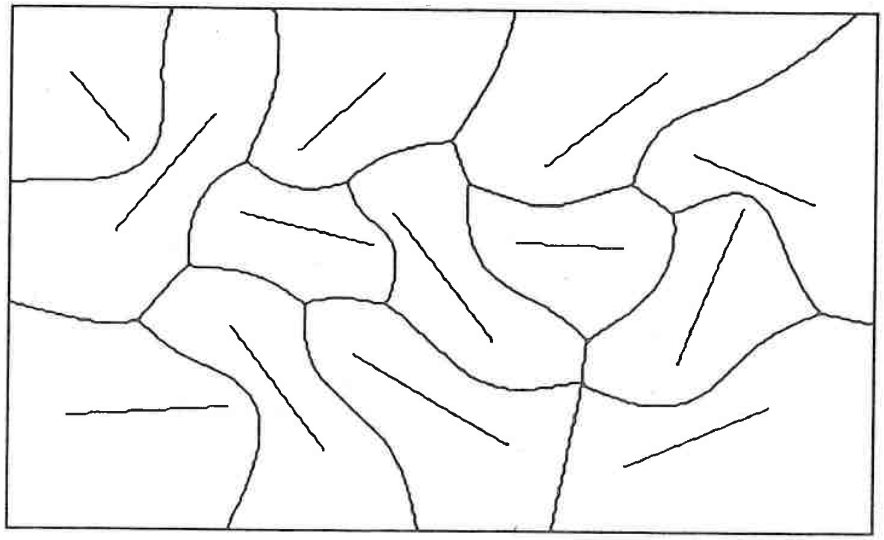


Figure 18 : Diagramme de Voronoï discret de segments

Remarque Il se peut, dans le cas d'un polygone englobant d'autres polygones et si la numérotation affecte à tous les arcs du polygone englobant un même numéro, que le diagramme ne soit pas connecté. Voir figure 18. Dans ce cas, l'algorithme devra trouver une cellule germe pour chacune des composantes connexes du diagramme.

3.3.5 Implantation de l'algorithme de calcul du diagramme de Voronoï discret d'un ensemble de polygones sur une machine parallèle : le T-Node

Ce travail a été réalisé par Alain Filbois sur la machine T-Node à 32 transputers ([A & F 90]). Cette machine extraordinaire préfigure les machines du futur. Elle est de type MIMD (multiple instructions, multiple data). Chaque transputer (microprocesseur connecté à d'autres microprocesseurs) possède 4 liens (nord, sud, est, ouest) avec ses transputers voisins (connexions hardware). Il est par contre possible de définir dynamiquement des liens logiques entre les transputers. Cette définition dynamique est propre au problème, elle est à la charge du programmeur et est réalisée par le dispositif appelé switcher. Chaque transputer possède sa

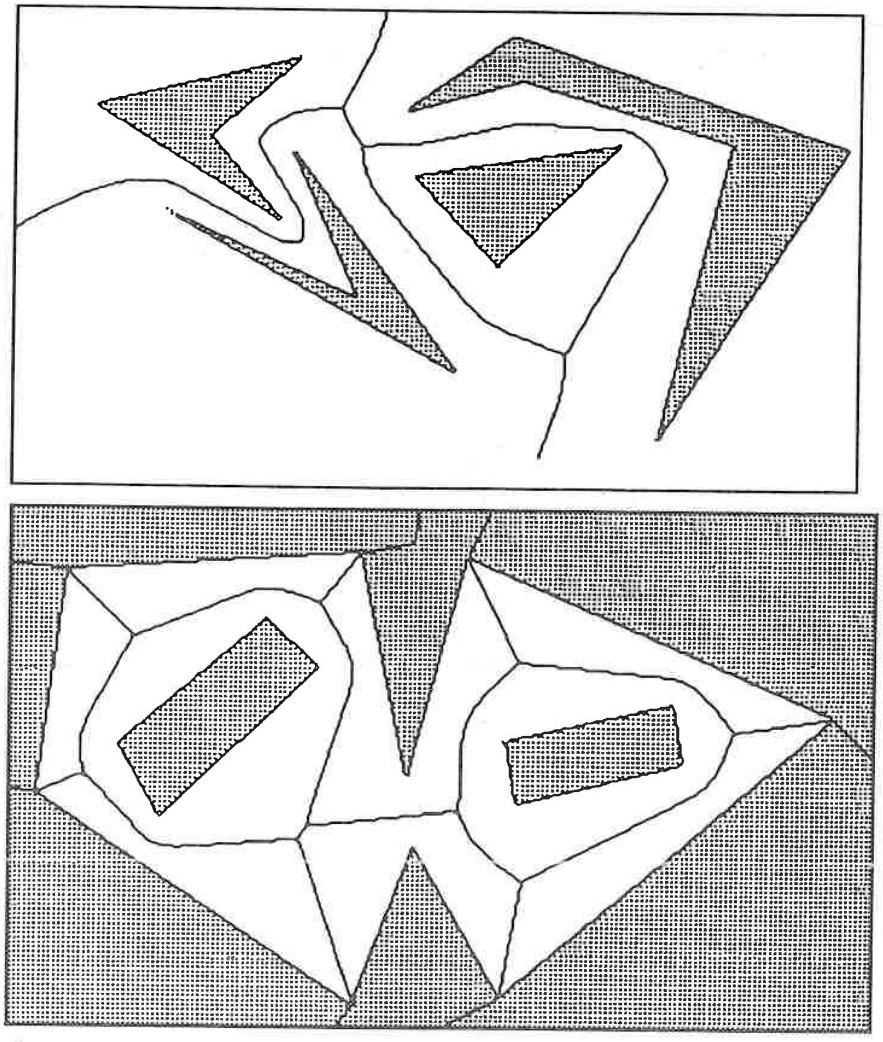


Figure 19 : Diagrammes de Voronoï discrets de polygones

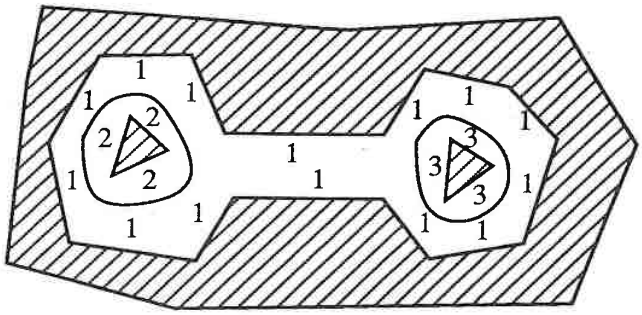


Figure 20 : Diagramme non connecté

mémoire propre. Un transputer particulier (contrôleur) possède une mémoire plus importante. Elle joue en quelque sorte le rôle d'une mémoire globale mais il faut bien comprendre que les transputers ne se "servent" pas dans une même mémoire commune. Tous les algorithmes proposés aujourd'hui travaillent sur une machine parallèle idéale qui n'existe pas.

Le parallélisme apparaît à 2 niveaux dans l'algorithme :

- Au niveau de l'examen des 8 voisins de la dernière cellule trouvée. Il est possible de réaliser cet examen en parallèle.
- Au niveau des arcs du diagramme. Lorsqu'un sommet du diagramme est atteint, il est possible de lancer 2 processus en parallèle sur les 2 arcs adjacents.

On appellera unité de recherche le groupement de transputers capables d'examiner les 8 voisins de la dernière cellule trouvée. Après une optimisation poussée, il apparaît qu'il est possible de réduire l'unité à 5 transputers. Elle sera constituée d'un maître et de 4 esclaves. Ceux-ci attendent en permanence que le maître leur donne un point pour lui rendre le plus proche segment de ce point. Pour des raisons physiques (4 liens par transputer), un seul de ces esclaves sera relié au maître. Il sera appelé contremaître et il sera relié aux 3 esclaves. Cette unité de recherche sera baptisée grappe. Le T-Node possédant 32 transputers, 6 grappes pourront être crées.
Le contrôleur aura pour charge de collecter les points reçus des grappes, d'empiler les débuts des nouveaux arcs de Voronoï trouvés et de les dépiler pour des grappes ayant terminé leur travail (sous-entendu, elles calculent des arcs déja trouvés ou bien elles ont atteint une concavité).
Une configuration logique possible d'implantation sera un arbre binaire, de racine le contrôleur et ayant pour noeuds internes ou feuilles les grappes. Le contrôleur sera relié au processeur d'interfacage avec l'extérieur. L'implantation peut être schématisée par la figure 21.

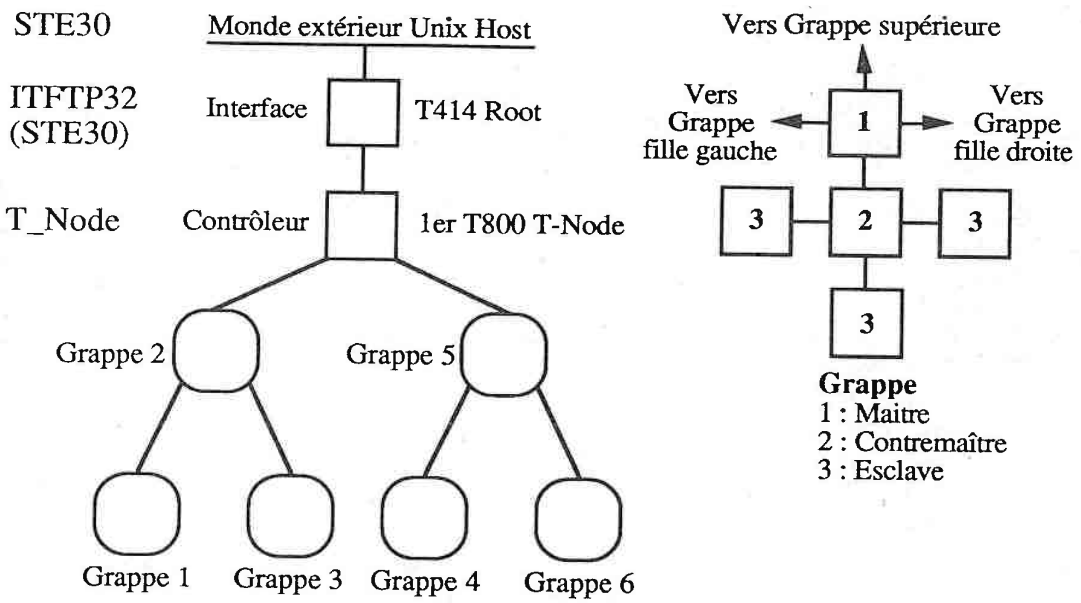


Figure 21 : Réseau des transputers et schéma d'une grappe

Les figures 22 et 23 donnent 2 exemples d'exécution. L'amélioration de performances a été tout à fait remarquable puisque le rapport entre les temps d'exécution sur une machine dotée d'un 68000 simple et le T-Node est supérieur à 500 ...

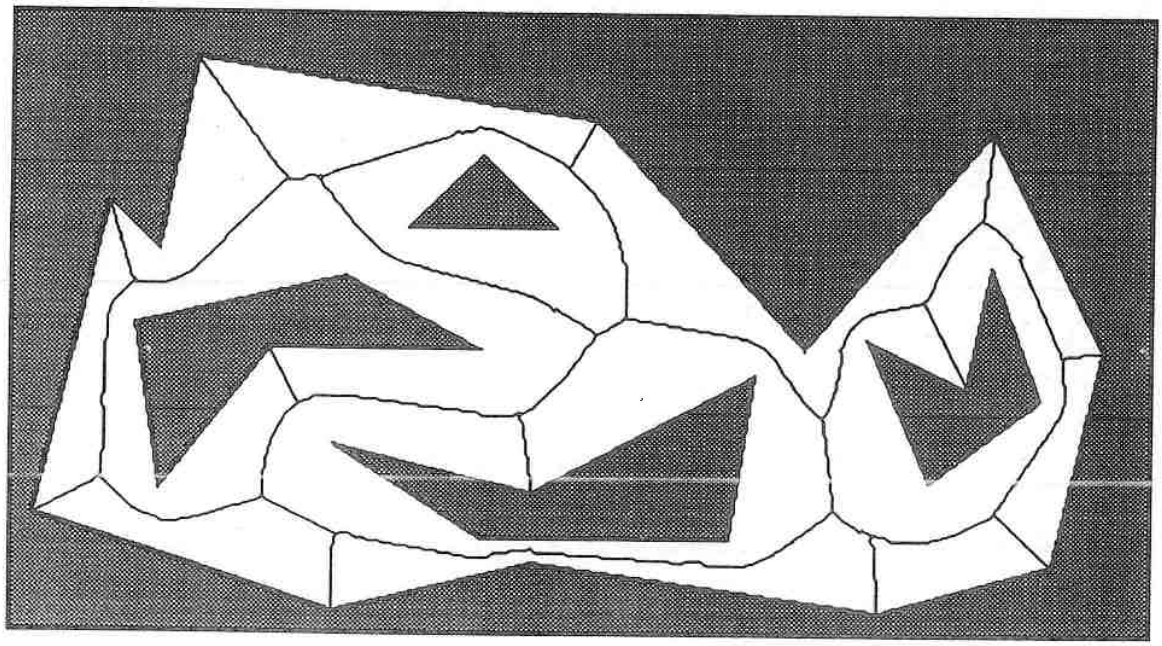


Figure 22 : Diagramme de Voronoï parallèle

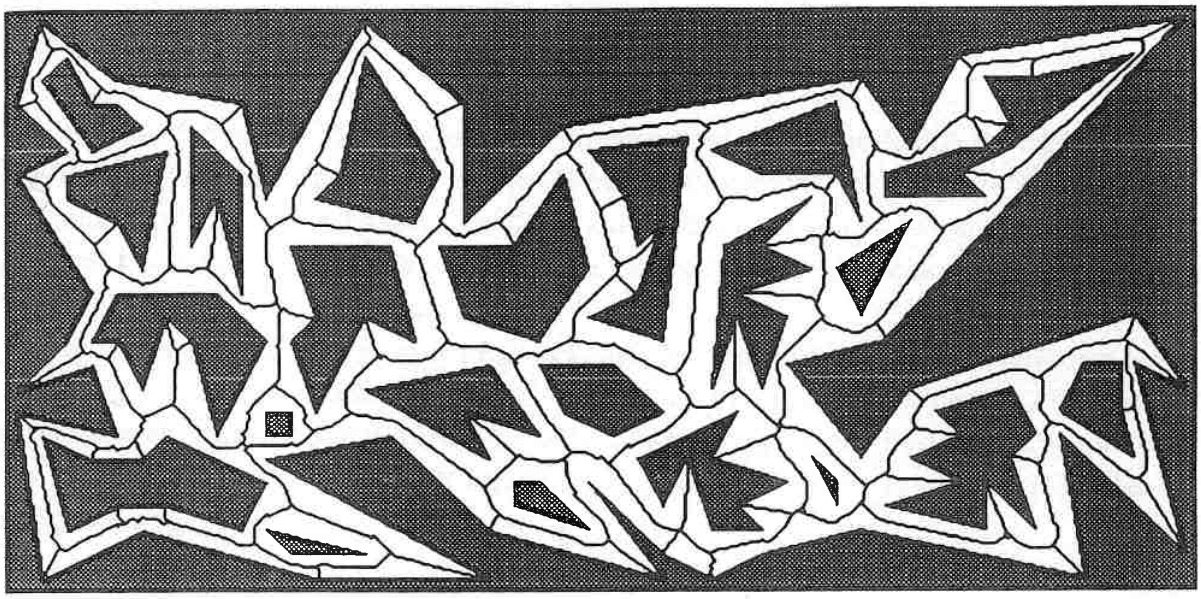


Figure 23 : Diagramme de Voronoï parallèle

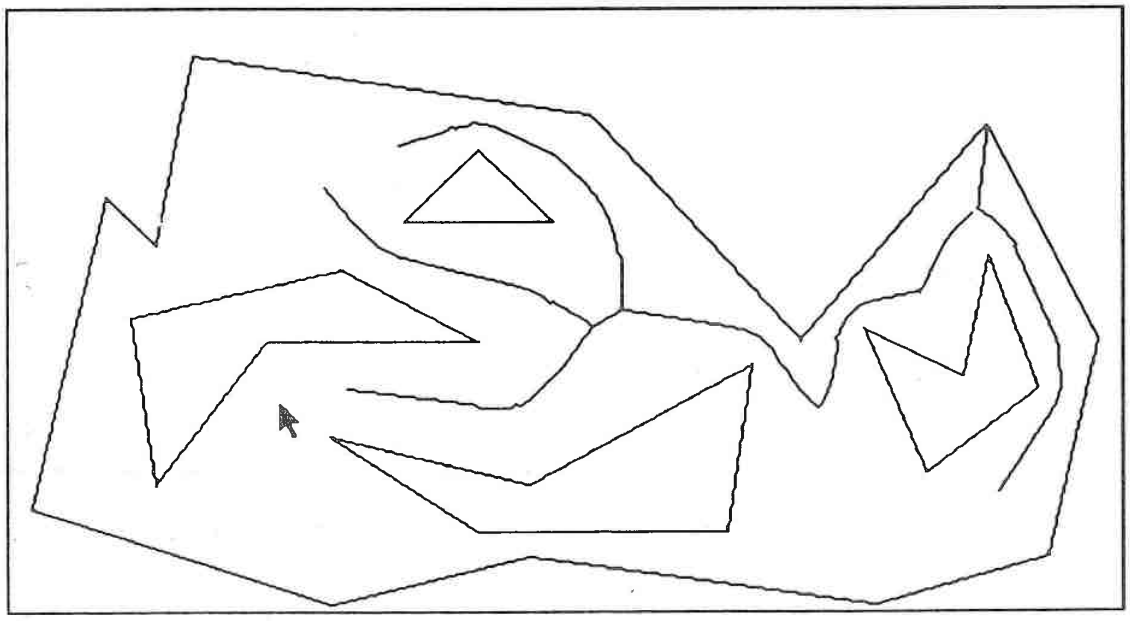


Figure 24: Diagramme de Voronoï parallèle en cours de constitution

3.4 Calcul discret du diagramme de Voronoï pour un polygone, pour un ensemble de polygones

Les définitions que nous avons données jusqu'ici étaient relatives à des points. Il s'agissait d'étudier la topologie de l'ensemble des **points** plus près de tel(s) objet(s) que de tel(s) autres. Il est tout à fait possible d'étendre ces définitions pour non plus des points mais un ou des objets

quelconques. Si nous considérons désormais un certain polygone P, la définition du diagramme devient :

Définition Soient P_i et P_j 2 polygones. La distance entre P_i et P_j, notée $d(P_i,P_j)$ est égale au min des $d(q,q')$ $q \in P_i$ et $q' \in P_j$. La fonction de base de l'algorithme calculant cette distance est celle calculant la plus petite distance entre 2 segments. Une analyse par cas, suivant l'appartenance des sommets de l'un aux régions V(a), V([a,b], ou V(b) de l'autre, résoud ce problème.

Soit S un ensemble de polygones $P_1,P_2,...,P_k$. Soit I un polygone de point de référence I_0. Notons $I_{x,y}$ le positionnement de I associé au point x,y (I_0 et (x,y) coincident). Le bord du diagramme de Voronoï, noté @Vor(S,I)), de $P_1,...,P_k$ pour le polygone I est égal à l'ensemble des points (x,y) tels qu'il existe P_i,P_j, $i \neq j$, $d(I_{(x,y)},P_i) = d(I_{(x,y)},P_j)$ et P_i, P_j sont 2 plus proches polygones de $I_{(x,y)}$.

Remarque A l'opposé du diagramme de Voronoï pour un point, ce nouveau diagramme n'est pas forcément connexe. Il y a autant de composantes connexes du diagramme que de composantes connexes de $P(I,S,\alpha)$ où α est la rotation du polygone I. On trouve ici un lien avec la notion d'espace libre définie au chapitre 2.

Supposition Nous nous placerons dans la même situation que celle définie au chapitre 2. Nous supposerons que l'espace libre est borné par l'existence d'un polygone englobant.

Définition Soit $P(I,S,\alpha)$ l'espace libre pour I au sein des polygones de S pour la rotation α fixée de I. Le diagramme de Voronoï de S pour I est égal au diagramme de Voronoï simple (relatif à un point et non à un polygone) des polygones de l'espace libre.

Preuve Soit Cp1 un polygone de $P(I,S,\alpha)$ et soit e un arc du diagramme de Voronoï de Cp1. Soit (x,y) un point de e. Par définition (x,y) est à équidistance de 2 points (x_1,y_1) et (x_2,y_2) appartenant au bord de Cp1 et ces 2 points sont les plus proches points de (x,y) vérifiant ces propriétés. Donc (x,y) est un point tel que les 2 plus proches contacts de $I_{(x,y)}$ sont équidistants donc (x,y) appartient bien au diagramme de Voronoï recherché (voir figure 25).

Le calcul du diagramme de Voronoï pour un polygone se ramène donc au calcul du diagramme pour un point sur les polygones de l'espace libre. La méthode décrite dans les paragraphes précédents le permettra. Il suffit d'affecter des numéros différents à tous les arcs de l'espace libre.

Complexité La meilleure complexité de l'algorithme de calcul de l'espace libre est $O(m^2.n^2)$ où m est le nombre d'arcs du polygone I et où n celui des polygones (voir [A & B 88]). Le nombre d'arcs de l'espace libre est en $O(m^2.n^2)$. Soit K le nombre de cellules du diagramme. La complexité globale est donc en $K*O(m^2.n^2)$ dans le pire cas.

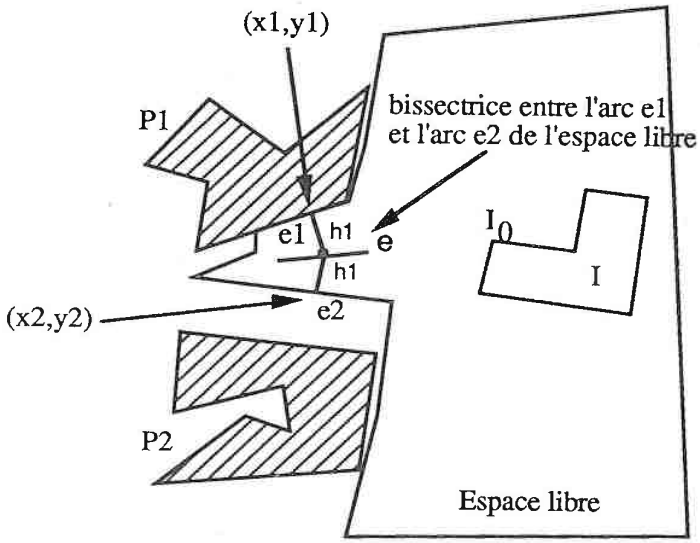


Figure 25 : Diagramme de Voronoï et espace libre

La figure 26 donne l'exemple d'un tel diagramme.

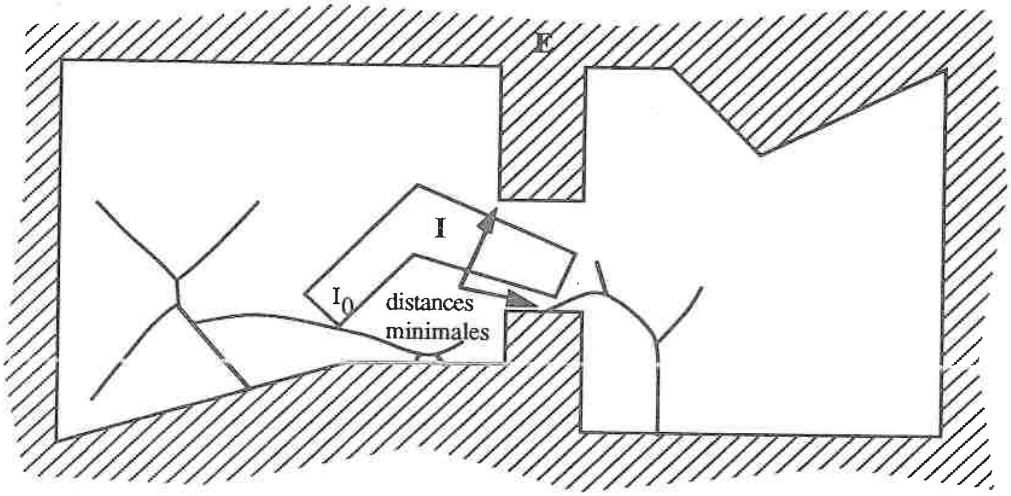


Figure 26 : Diagramme de Voronoï pour un polygone

3.5 Conclusion sur les algorithmes de calcul des diagrammes de Voronoï

Nous avons présenté quelques algorithmes calculant des diagrammes différents dans des espaces différents. Ils seront un outil puissant pour la planification de trajectoires. Nous proposons au chapitre suivant une utilisation concrète de ces diagrammes.

Chapitre 4

Discrétisation et heuristiques pour résoudre le déplacement en translation et rotation d'un polygone

4.1 Introduction

La méthode proposée au chapitre 2 ne possède qu'une qualité : celle d'être exacte. Ses inconvénients sont par contre nombreux :

• Elle est très peu dynamique : toute modification de l'environnement est certainement difficile à prendre en compte au niveau du modèle.sans avoir à recalculer tout.

• Elle est certainement peu robuste. Les techniques d'analyse numérique résolvant les équations sont fragiles. Les algorithmes de type plane-sweep manipulant des segments le sont aussi.

• Elle est inutilement puissante : en particulier, un déplacement très localisé entraîne le calcul de l'espace libre tout entier alors que seule une petite portion de celui-ci intéresse le robot. De plus, la parfaite connaissance de la topologie de l'espace libre est dans la plupart des cas, inutile.

• Les déplacements sont recherchés sur le bord de l'espace libre, donc en contact avec l'environnement , ce qui n'est pas réaliste.

Nous allons proposer dans ce chapitre ainsi qu'au suivant , une nouvelle orientation. L'univers de travail sera un univers discret. Il s'avére que dans bon nombre de cas, les opérations classiques de la géométrie algorithmique deviennent très simples. Il devrait être facile de concevoir des architectures spécialisées ainsi que de parallèliser les algorithmes.

La méthode que nous allons proposer s'inspire des méthodes de rétraction mais elle n'en est pas vraiment une. La rétraction utilisée sera, dans le cas du déplacement dans un couloir, une fonction spline et dans le cas du déplacement parmi un ensemble de polygones, le diagramme de Voronoï des polygones. Nous allons donner tout d'abord quelques définitions générales.

Définition Soit EC l'espace des configurations et soit k sa dimension. On définit une discrétisation notée CEL_{EC} de EC en cellules régulières. Une discrétisation est totalement définie par la donnée de h valant la taille des arêtes des cellules. Une cellule est définie par un k-uplet d'entiers. La cellule $(a_{i1},...,a_{ik})$ d'arête h vaut le sous-ensemble, noté $Cube_{(ai1,...,aik)}$, de $\mathbb{R}^k$ défini par :

$Cube_{(ai1,...,aik)} = \{(x_1,...,x_k), x_1,...,x_k \in \mathbb{R}, a_{ij}*h \leq x_j \leq (a_{ij} + 1)*h, j \in [1..k]$

On a donc une bijection entre CEL_{EC} et $\mathbb{Z}^k$.

Définition Deux cellules $(a_{i1},...,a_{ik})$ et $(a_{j1},...,a_{jk})$ sont voisines si $\forall\ t \in [1..k]\ |a_{it}-a_{jt}| \leq 1$. Dans $\mathbb{Z}^2$, une cellules peut avoir 8 voisins (on parle de 8-connexité) et 26 dans $\mathbb{Z}^3$ (voir figure 1).

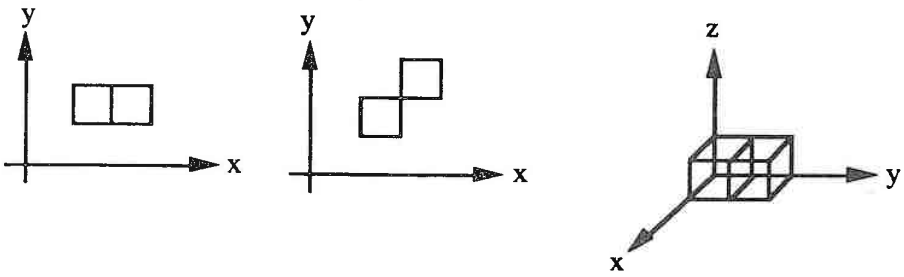


Figure 1 : Cellules voisines dans $\mathbb{Z}^2$ ou $\mathbb{Z}^3$

Définition Une trajectoire discrète, notée T_{EC} sera définie comme une suite de cellules voisines. On a : $T_{EC} = \{(a_{11},...,a_{1k}) , (a_{21},...,a_{2k}),...,(a_{p1},...,a_{pk})\}$ où $(a_{i1},...,a_{ik})$ voisin de $(a_{i+1\ 1},...,a_{i+1\ k})$ pour $i \in [1..p-1]$. p sera appelé longueur de la trajectoire.

Définition Soit $(a_{i1},...,a_{ik})$ une cellule de CEL_{EC}. On appellera positionnement de I suivant la configuration $(a_{i1},...,a_{ik})$, noté $P_{(ai1,...,aik)}(I)$, la position physique de I suivant la configuration $(a_{i1}*h+h/2,...,a_{ik}*h+h/2)$. Autrement dit, I est positionné suivant le centre de la cellule $(a_{i1},...,a_{ik})$.

Définition On dira que la trajectoire $T_{EC} = \{(a_{11},...,a_{1k}) , (a_{21},...,a_{2k}),...,(a_{p1},...,a_{pk})\}$ est libre si pour toute cellule $(a_{i1},...,a_{ik})$ de T_{EC} on a $P_{(ai1,...,aik)}(I) \cap E = \varnothing$

Conclusion Nous avons défini brièvement l'univers discret dans lequel évolue le système robotique I. Nous avons défini ce qu'est une trajectoire discrète libre. Nous faisons désormais la supposition essentielle suivante : si T_{EC} est une trajectoire discrète libre, alors nous supposerons qu'il existe une trajectoire continue libre reliant les centres des cellules de T_{EC}. Cette supposition devient vraie quand $h \rightarrow 0$. Soient C_1 et C_2 deux cellules voisines dans l'espace des configurations. La trajectoire continue entre ces 2 cellules est obtenue en faisant varier linéairement les paramètres de configuration entre les centres des 2 cellules. Le choix de h sera guidé par la nature de l'environnement ainsi que par l'expérience.

4.2 Le principe de la méthode : La recherche guidée

4.2.1 Introduction

On allons tout d'abord donner un algorithme naïf de recherche d'une trajectoire dans l'espace discret des configurations. Son principe est particulièrement simple. Soient $(a_{dep1},...,a_{depk})$ (C_{dep}

en abrégé) et (a_{fin1},...,a_{fink}) (C_{fin} en abrégé) les configurations de départ et d'arrivée de la trajectoire. Une simple recherche en largeur d'abord dans le graphe de connexité des cellules (notion de voisinage) permet d'exhiber une trajectoire. On supposera, pour la cellule courante (a_{i1},...,a_{ik}), que l'on dispose d'un algorithme qui dit s'il existe une intersection entre $P_{(ai1,...,aik)}(I)$ et l'environnement E.

L'algorithme (connu sous le nom A*) maintient une liste de chemins où chaque noeud n est affecté d'une valuation f(n) somme de g(n), meilleur coût pour atteindre n depuis la configuration origine, et h(n) coût heuristique évaluant le coût du chemin restant à accomplir jusqu'à la configuration terminale. h(n) peut valoir la distance entre n et la configuration terminale. Le coût relatif au passage d'une cellule n à une de ses voisines n' (noté C(n,n')) vaudra une constante.

Une étape de l'algorithme consiste à mettre à jour 2 listes de noeuds du graphe. La liste O des noeuds atteints mais dont on n'a pas examiné tous les voisins et la liste F des noeuds dont on a examiné tous les voisins. On maintient pour chaque noeud examiné un pointeur vers son prédécesseur sur le meilleur chemin qui y conduit.

On a :

En entrée : C_{dep} et C_{fin}

En sortie : une trajectoire libre si elle existe.

début

- *Initialiser O avec C_{dep} et calculer pour ce noeud $g(C_{dep})$ et $h(C_{dep})$*
- *Initialiser F avec $\varnothing$*
- *Tant que $O \neq \varnothing$ répéter*
    - *choisir dans O le noeud n tel que g(n) + h(n) minimum*
    - *si $n = C_{fin}$ alors retourner(succés)*
    - *pour chaque voisin n' de n tel que $P_n(I) \cap E = \varnothing$ répéter*
        - *$f' = g(n) + coût(n,n') + h(n')$ /* coût associé au noeud n' */*
        - *si $n' \notin O$ et si $n' \notin F$*
            - *alors* • *ajouter n' à O avec $g(n') = g(n) + coût(n,n')$*
              • *attacher un pointeur de n' à n*
            - *sinon* • *si $f' < g(n') + h(n')$*
                - *alors* • *$g(n') = g(n) + coût(n,n')$*
                  • *modifier le pointeur de n' par un pointeur sur n*
                  • *si $n' \in F$ alors le déplacer dans O*

    fin pour
    - *Déplacer n de O dans F*

fin tant que

fin

Cet algorithme naïf (mais exact à la discrétisation près) appelle plusieurs remarques :

• La complexité inhérente à la planification de trajectoires joue ici à fond. Le nombre de cellules est une fonction exponentielle du nombre de degrés de liberté k. Donc, cet algorithme d'exploration a une complexité exponentielle.
• Le robot ne posséde qu'une vue très locale de l'environnement. Le seul critère lui permettant de ne pas errer est la fonction heuristique h qui l'incite à se rapprocher du but. Il se peut qu'il perde beaucoup de temps dans les parties concaves des obstacles.
• Certaines portions de trajectoires seront en contact avec l'environnement ce qui dans la pratique n'est pas souhaitable.

Nous allons proposer un algorithme qui tente d'apporter une solution aux remarques précédentes :
• La complexité de l'algorithme sera diminuée : la recherche de la suite connexe de cellules libres dans l'espace des configurations sera guidée au point de devenir linéaire. Nous entendons par là que le nombre de voisins de la cellule courante sera réduit à 1.
• Les concavités des obstacles seront évitées de par la nature même de la trajectoire guide : En effet, nous avons retenu comme trajectoire guide le diagramme de Voronoï (ou une trajectoire très approchante) de l'environnement.
• Les déplacements seront le plus possible éloignés des obstacles.

Par contre, la qualité d'exactitude (à la discrétisation près) sera perdue. Nous allons préciser maintenant les idées de base concernant la recherche guidée.

4.2.2 La trajectoire guide

Cette trajectoire guide sera en fait un graphe. Il sera noté G(E) où E est l'environnement. Ce graphe sera construit de telle façon que pour chacun de ses points (i,j) il n'existe pas de point (k,t) du plan qui soit plus "éloigné" des obstacles que (i,j) et qui n'appartienne pas au graphe G(E). L'éloignement entre un point et un ensemble d'obstacles représente la plus petite distance qui peut exister entre ce point et un point quelconque d'un obstacle. Le lecteur aura déja remarqué que ce graphe est le graphe de Voronoï des obstacles. Nous considérerons dans un premier temps que la trajectoire guide est une portion quelconque de trajectoire dicrète.

Donnons maintenant quelques définitions concernant le déplacement de I sur la trajectoire guide :

Définition Le système robotique I sera composé d'un polygone simple. Soit [LT,RT,RB,LB] le rectangle englobant I et tel que le segment [LT,RT] soit de longueur minimale. Soit xOy un

repère attaché à I, de centre le centre du rectangle englobant et dont les axes sont parallèles aux cotés du rectangle (voir figure 2)

Définition Soit T un point défini dans xOy. Il sera initialisé au milieu du segment [LT,RT]. Il jouera un rôle crucial dans le déplacement. Soit Q un point appartenant à la trajectoire guide et situé à une distance F du point T. Le positionnement courant du polygone I est réalisé comme suit : soit (i,j) le point courant sur la trajectoire guide. I est positionné sur celle-ci de telle façon que le point T coïncide avec (i,j) et que l'axe des y soit parallèle au segment [T,Q] (voir figure 3). On dira que T et F sont les paramètres de positionnement de I sur la trajectoire guide.

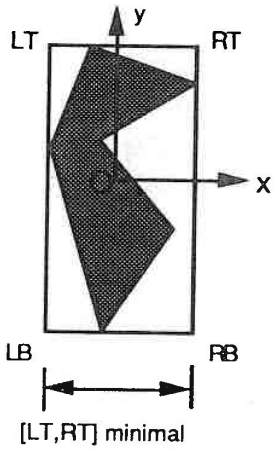


Figure 2 : Définition
du rectangle englobant

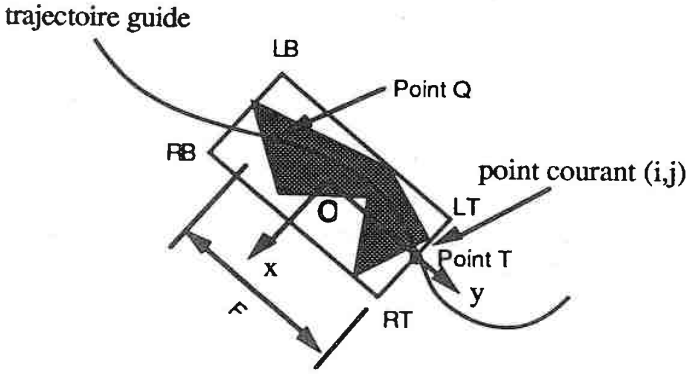


Figure 3 : Positionnement de I
sur la trajectoire guide

Nous pouvons donner l'algorithme général de déplacement sur la trajectoire guide discrète :

Début

Tant que non(extrémité) Répéter

- *soit (n_x,n_y,n_{alpha}) la configuration actuelle du robot (n_x,n_y,n_{alpha} sont des entiers relatifs)*
- *Déplacer le point courant (i,j) sur la cellule suivante de la trajectoire discrète*
- *Déterminer le point Q situé à la distance F de (i,j)*
- *soit (m_x,m_y,m_{alpha}) la configuration du robot relative au positionnement de I à l'aide des paramètres T et F.*
- *créer le segment discret entre (n_x,n_y,n_{alpha}) et (m_x,m_y,m_{alpha}) (algorithme de Brésenham dans l'espace des configurations) et effectuer le déplacement du robot sur la suite des positions élémentaires voisines sur le segment.*

fin tant que

fin

Conclusion Nous avons proposé une méthode très simple permettant au robot I de suivre une certaine trajectoire guide. Il n'a pas été question pour l'instant de détection de collision ni de la conception de la trajectoire. Remarquons qu'elle est indépendante de la géométrie du robot et qu'elle peut donc être établie une fois pour toute sur l'environnement. Notons enfin que nous faisons la supposition que si le positionnement de I suivant (n_x,n_y,n_{alpha}) est libre et le positionnement de I suivant (m_x,m_y,m_{alpha}) est libre lui aussi alors les cellules discrètes situées sur le segment $[(n_x,n_y,n_{alpha}) , (m_x,m_y,m_{alpha})]$ le sont aussi.

4.3 Détection et résolution des collisions

Nous allons préciser ici les techniques de détection des collisions ainsi que les procédures mises en oeuvre pour y remédier. Précisons immédiatement qu'elles sont très simples, pour ne pas dire grossières, et qu'il est tout à fait possible de les améliorer.
Notons tout d'abord que les collisions se produisent dans les parties courbes de la trajectoire guide (en effet, si la trajectoire guide est rectiligne, alors le déplacement est rectiligne et la technique de positionnement à l'aide des paramètres T et F devient exacte). Nous avons établi 3 types de collisions. Ce sont :

- Les collisions de type "bas" (voir figure 4 à gauche). Intuitivement, elles se produisent dans la partie interne d'une portion courbe de la trajectoire guide.
- Les collisions de type "queue" (voir figure 4 au milieu). Intuitivement, elles se produisent dans la partie externe d'une portion courbe de la trajectoire guide.
- Les collisions de type "étranglement" (voir figure 4 à droite). Elles combinent les 2 catégories précédentes.

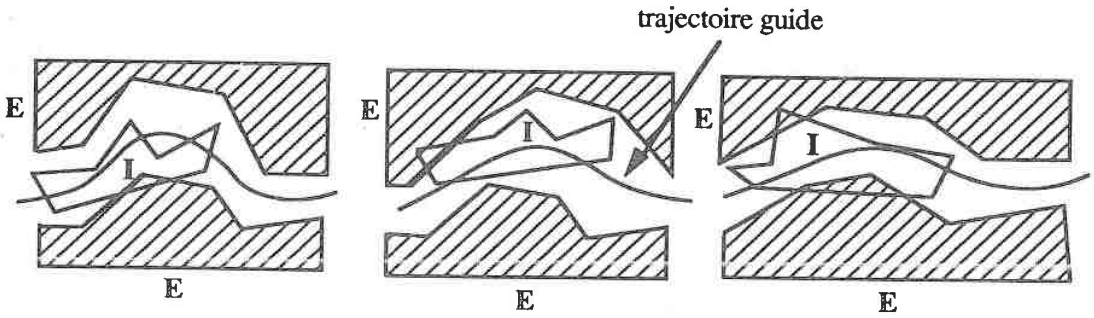


Figure 4 : Les différents types de collision : "bas", "queue" et "étranglement"

Résolution des collisions
Par résolution des collisions, nous entendons les modifications à apporter aux paramètres de positionnement pour que la collision n'ait plus lieu.

Résolution des collisions de type "bas"

Nous proposons 2 méthodes. La première consiste à réduire F tout en laissant le point T fixe dans le repère xOy. Cette réduction a pour effet d'engendrer une rotation qui écarte I de la zone interne de la courbe de la trajectoire guide et par la même, de l'obstacle qui causait la collision (voir figure 5 à gauche). La seconde méthode consiste à déplacer T vers le point RT tout en laissant F fixe. Ce déplacement aura aussi pour effet d'écarter I de la zone de collision (voir figure 5 à droite).

Résolution des collisions de type "queue"

Les méthodes à appliquer sont inverses des précédentes. L'augmentation de F aura pour effet d'engendrer une rotation qui rapproche I de la zone interne de la courbe et l'écarte ainsi de la zone de collision (voir figure 6 à gauche). Le déplacement de T en direction du point LT aura pour effet d'écarter I de la zone externe de la courbe (voir figure 6 à droite).

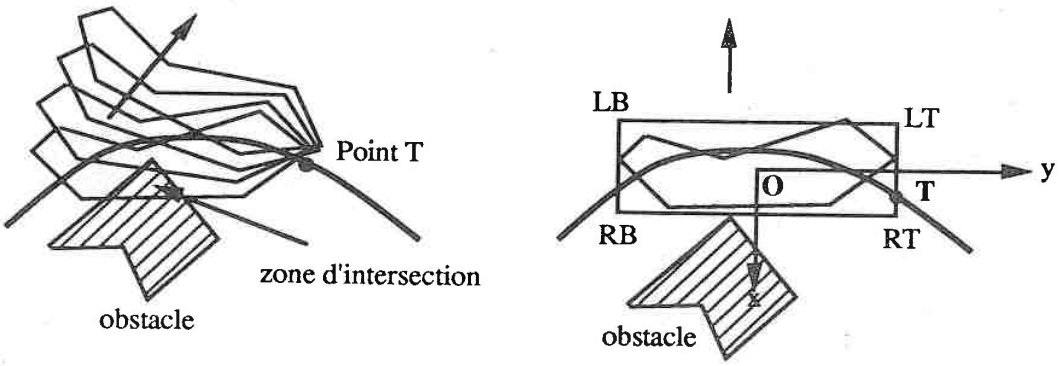


figure 5 : Résolution des collisions de type "bas"

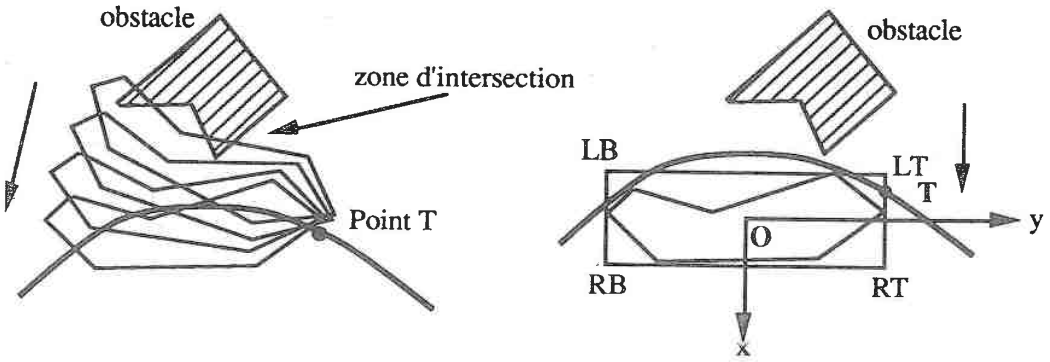


figure 6 : Résolution des collisions de type "queue"

Résolution des collisions de type "étranglement"

Nous n'avons pas implanté d'heuristiques pour la résolution de ce type de collision évidement plus difficile à résoudre Dans ce cas, la trajectoire est déclarée impossible et le planificateur global en propose une autre.

Détection des collisions

Nous n'avons retenu qu'une approche naïve (mais efficace vu les outils employés) pour calculer les intersections entre le robot et son environnement. Cet algorithme sera mis en jeu pour chaque position élémentaire du robot. Il est de type bitmap. Nous reviendrons plus en détail dans un prochain chapitre sur ce type d'algorithme. Disons simplement que les polygones (l'environnement et le robot) sont représentés dans des matrices booléennes et qu'il suffit de réaliser le "et" logique entre les 2 matrices pour obtenir le polygone résultat (voir figure 7). Notons aussi que cet algorthme est local. Seule la portion de l'environnement localisée dans le rectangle englobant du robot sera manipulée pour le calcul de l'intersection.

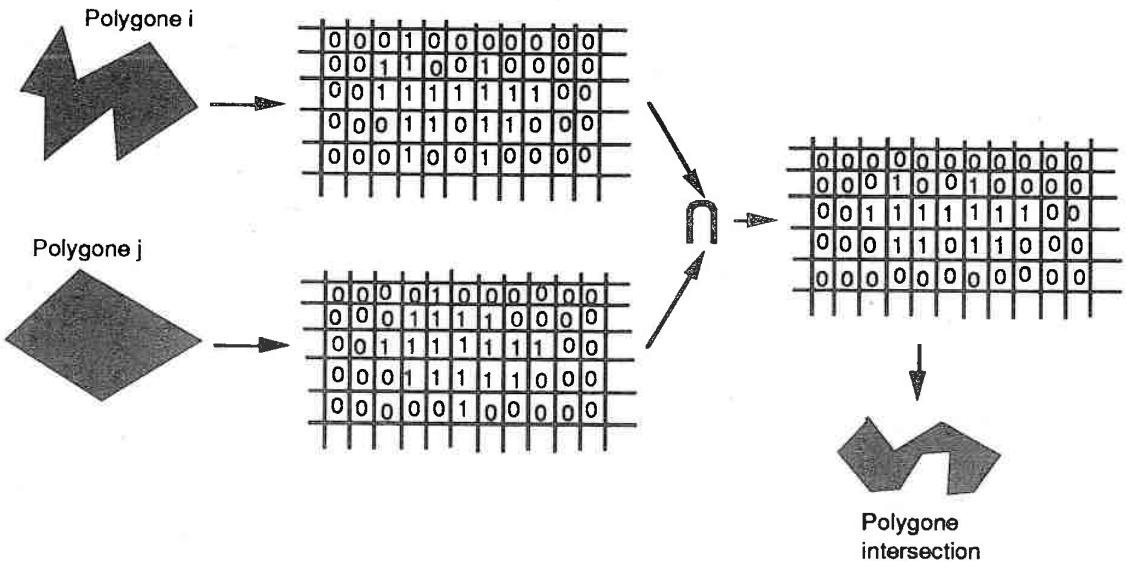


Figure 7 : Intersection de 2 polygones bitmap

Nous sommes en mesure de donner un algorithme général de déplacement du robot I le long d'une trajectoire discrète guide, tout en tenant compte des collisions éventuelles ainsi que de leur résolution.

Convention Dans le cas où une collision est détectée, on appelera Résol(T,F) la procédure qui fait varier, d'un certain pas discret élémentaire, les 2 paramètres T et F.

En entrée : ◦ Une trajectoire guide discrète

- L'environnement E
- Le robot I

En sortie : • Une trajectoire discrète libre, dans l'espace des configurations (si elle existe).

Début

Tant que non(extrémité) répéter

- *soit (n_x,n_y,n_{alpha}) la configuration actuelle du robot*
- *Déplacer le point courant (i,j) sur la cellule suivante de la trajectoire discrète*
- *Déterminer le point Q situé à la distance F de (i,j)*
- *soit (m_x,m_y,m_{alpha}) la configuration du robot relative au positionnement de I à l'aide des paramètres T et F.*
- *si $P_{(mx,my,malpha)}(I) \cap E \neq \varnothing$*

alors si collision de type "bas"

alors
- *Choisir une méthode (réduire F ou déplacer T)*
- *Tant que collision de type "bas" répéter*
  - *Résol(T,F)*
  - *soit (m_x,m_y,m_{alpha}) la configuration du robot relative au positionnement de I à l'aide des paramètres T et F.*
  - *calculer l'intersection entre $P_{(mx,my,malpha)}(I)$ et E*

fin tant que

sinon si collision de type "queue"

alors
- *Choisir une méthode (réduire F ou déplacer T)*
- *Tant que collision de type "queue" répéter*
  - *Résol(T,F)*
  - *soit (m_x,m_y,m_{alpha}) la configuration du robot relative au positionnement de I à l'aide des paramètres T et F*
  - *calculer l'intersection entre $P_{(mx,my,malpha)}(I)$ et E*

fin tant que

sinon
- *retourner(échec)*

fsi

fsi

sinon
- *créer le segment discret entre (n_x,n_y,n_{alpha}) et (m_x,m_y,m_{alpha}) (algorithme de Brésenham) et effectuer le déplacement du robot sur la suite des positions élémentaires voisines sur le segment.*

• *ajouter les positions élémentaires à la trajectoire résultat*

fsi
fin tant que
fin

Remarque Il se peut que la résolution d'une collision de type "bas" provoque une collision de type "queue" et vice-versa. Dans ce cas, si ce phénomène se reproduit un certain nombre de fois alors la trajectoire est déclarée impossible. De plus, si les paramètres de positionnement sortent d'une certaine norme (F devenant trop "grand" par exemple) alors la trajectoire est aussi déclarée impossible.

Conclusion Nous avons proposé un algorithme très simple de cheminement d'un polygone le long d'une trajectoire guide discrète. Il présente néanmoins l'inconvénient d'être très difficilement mesurable. Il n'est pas facile de déterminer les trajectoires qui lui échapperont. Il donne des résultats satisfaisants lorsque la forme du robot I est allongée et donne des résultats médiocres lorsque celle-ci est mal approximée par le rectangle englobant (cas d'une forme en boomerang).

4.4 Application de la méthode au déplacement dans un couloir quelconque

Définition Un couloir quelconque est obtenu par le rapprochement de 2 chaînes polygonales infinies (voir figure 8).

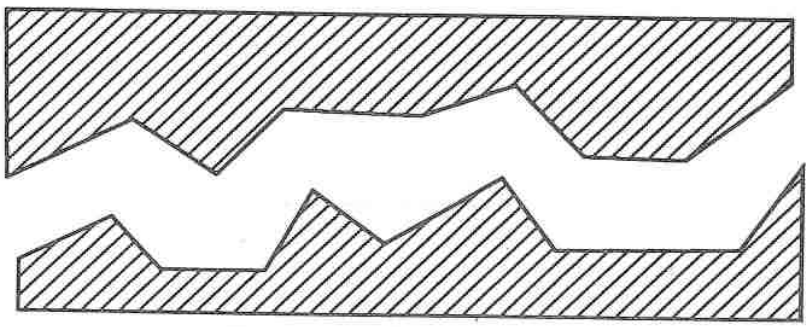


Figure 8 : Couloir quelconque

Constitution de la trajectoire guide

La trajectoire guide idéale est constituée par le diagramme de Voronoï des 2 bords du couloir quelconque. Nous proposons néanmoins une alternative dans cette étude. La trajectoire guide sera une fonction spline d'ordre 3 sur des points particuliers du couloir. Considérons la suite $t_1,t_2,...,t_n$ des sommets du bord supérieur ainsi que la suite $s_1,s_2,...,s_k$ des sommets du bord inférieur. Soit t'_i le point le plus proche du sommet t_i et situé sur le bord inférieur (t'_i n'est pas

forcément un sommet parmi les s_i). Soit s'_j le point le plus proche du sommet s_j et situé sur le bord supérieur. Les points particuliers du couloir seront constitués par la suite des milieux des segments $[t_i,t'_i]$ ou $[s_j,s'_j]$. La trajectoire guide sera la fonction spline d'ordre 3 reliant ces points. Nous avons retenu l'ordre 3 pour ses qualités "naturelles". La trajectoire guide n'est ni trop tendue ni trop sinueuse. Cette trajectoire peut ensuite être discrétisée de 2 façons différentes : une fonction spline étant paramètrée par t variant entre 0 et 1, on discrétise facilement en choisissant un certain pas sur t (0.01 par exemple). La seconde méthode consiste, à l'instar du diagramme de Voronoï discret, à calculer une suite connexe de cellules carrées intersectées par la fonction spline. Le calcul, pour un sommet donné, de son point le plus proche sur l'autre bord, se fait en calculant le plus proche point sur chaque segment puis en retenant le minimum Le calcul du plus proche point sur un segment [a,b] depuis le point P se fait de la façon suivante : soit Q la projection de P sur la droite (a,b). Si $Q \in [a,b]$ alors le plus proche point est Q sinon le plus proche point est soit a soit b.

La figure 9 donne un exemple de constitution de la trajectoire guide.

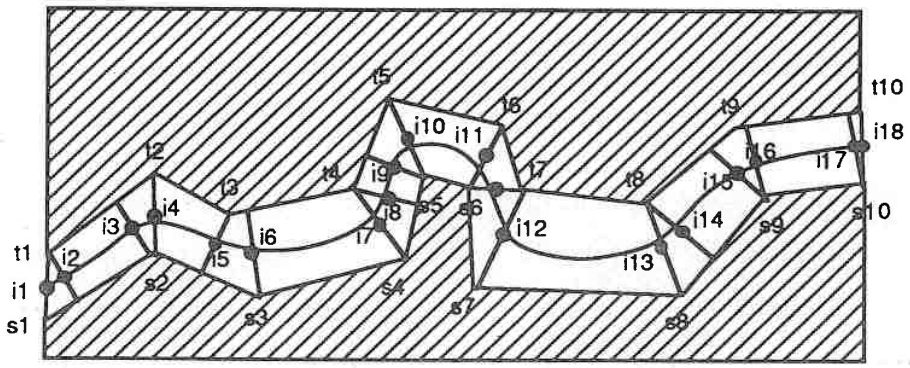


Figure 9 : Trajectoire guide dans un couloir quelconque

Conclusion Nous avons proposé une méthode simple de constitution d'une trajectoire guide. Notons que cette méthode est prise en défaut pour certaines configurations de couloirs quelconques (les points milieux des segments peuvent éventuellement appartenir au corps du couloir). Elle est par contre rapide. Le cheminement et la correction des collisions se font suivant les méthodes explicitées précédement. Les figures 10 et 11 donnent 2 exemples. Cette étude a donné lieu à un logiciel implanté en C. Le couloir ainsi que le polygone I sont définis interactivement. Le logiciel calcule la trajectoire guide puis effectue le déplacement. Notons que dans ce type de méthode, la réponse à la question "le déplacement est-il possible ?" est obtenue en même temps que le déplacement est effectué.

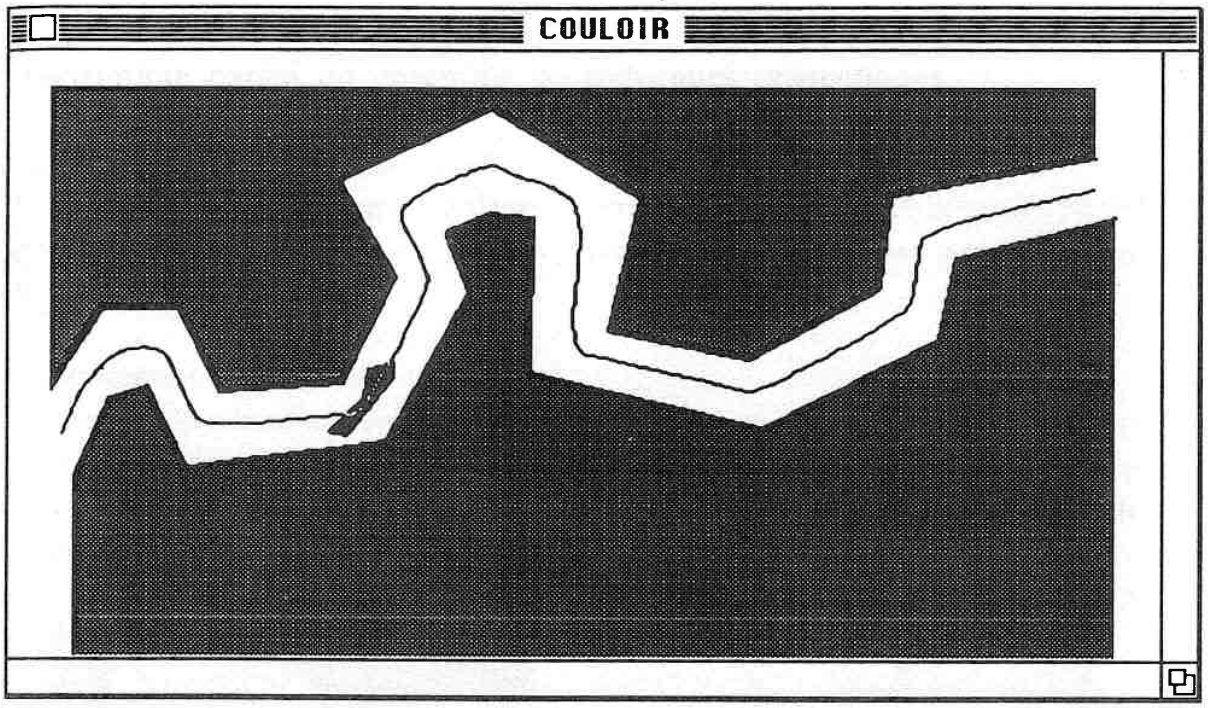


Figure 10 : Déplacement d'un polygone quelconque dans un couloir quelconque

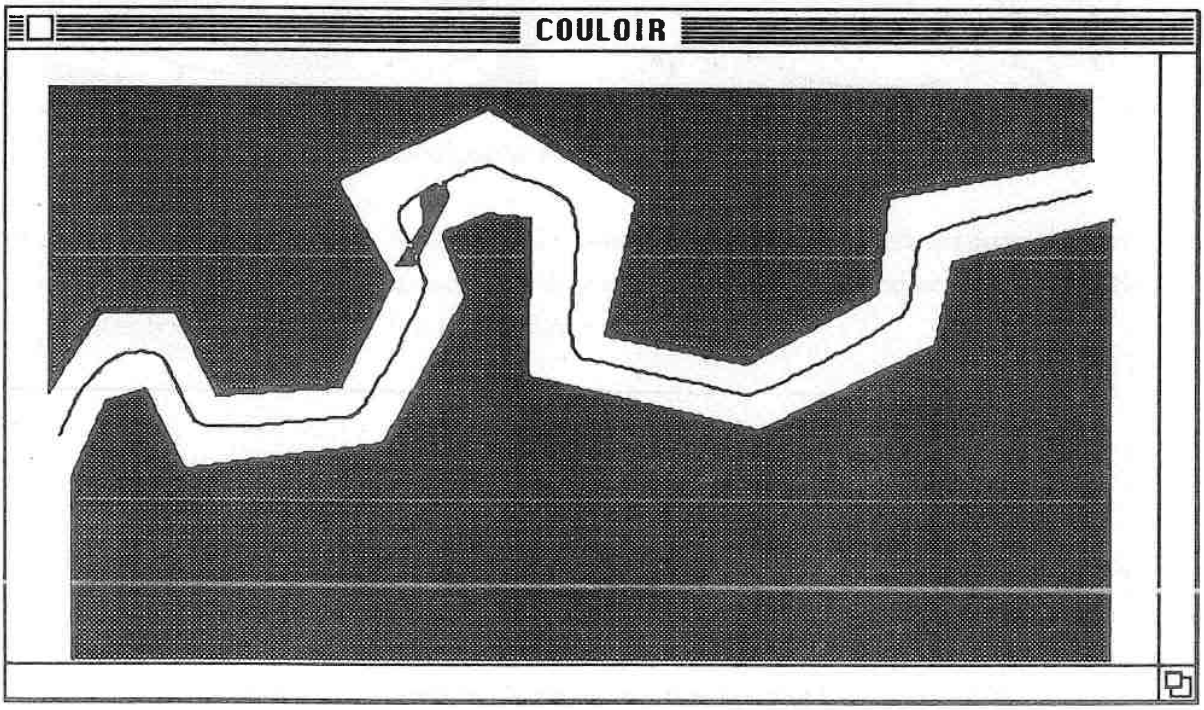


Figure 11 : Autre situation

4.5 Application de la méthode au déplacement d'un polygone quelconque parmi un ensemble de polygones quelconques

Constitution de la trajectoire guide

La trajectoire guide sera le graphe de Voronoï discret de l'ensemble des polygones. Il a été défini au chapitre 3 et nous ne reviendrons pas sur sa définition. Les caractéristiques propres à un graphe seront exploitées.

Prétraitement sur le graphe de Voronoï

Il consiste à localiser les noeuds puis à bâtir le graphe abstrait qui donne pour un noeud quelconque la liste de ses voisins ainsi que les valuations des arcs menant à ceux-ci. La valuation d'un arc sera égale au nombre de cellules de cet arc (voir figure 12). Elle donne une idée de la longueur d'un arc.

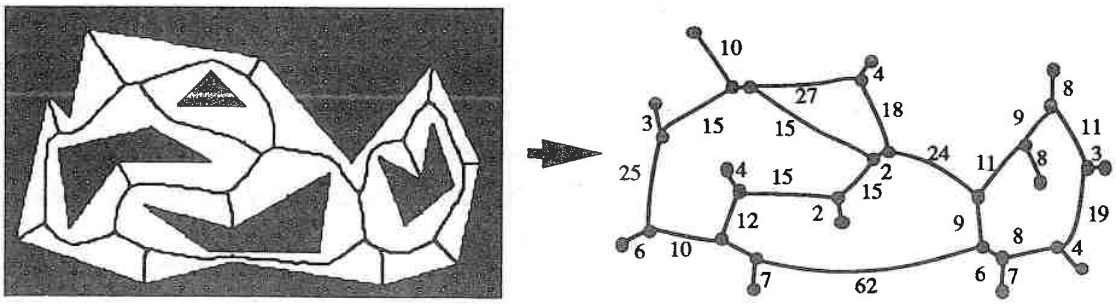


Figure 12 : Obtention du graphe abstrait à partir du diagramme de Voronoï

Restriction du problème : Nous faisons ici une restriction sur les configurations de départ et d'arrivée du robot : elles correspondront uniquement aux noeuds du graphe de Voronoï. Elles seront fonction des paramètres T et F (voir figure 13).

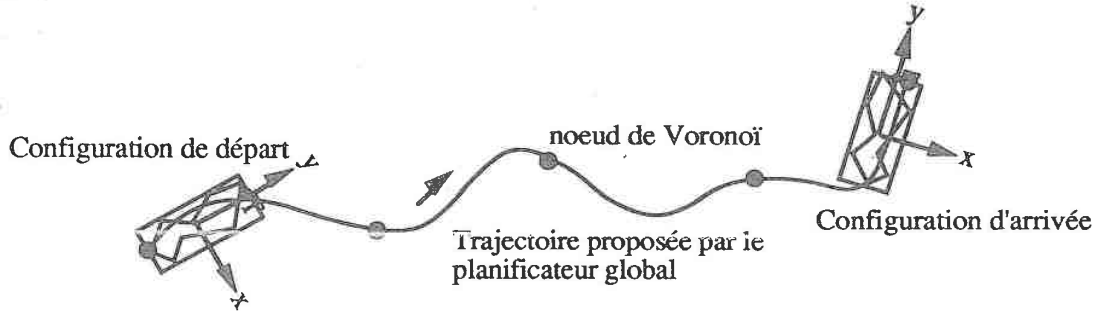


Figure 13 : Configurations de départ et d'arrivée sur la trajectoire guide

Nous n'avons pu étudier sérieusement, par manque de temps, le cas général. Il conviendrait dans un premier temps de localiser le robot dans le graphe de Voronoï puis de lui appliquer une

translation jusqu'à un certain point du graphe puis enfin appliquer la rotation correcte lui permettant un positionnement sur la portion de graphe correspondante.

Planification Le planificateur global effectue une recherche en largeur d'abord dans le graphe de Voronoï abstrait et propose ainsi une suite de trajectoires depuis la plus courte vers de plus longues. L'algorithme s'arrêtera lorsque le déplacement aura été possible sur l'une des trajectoires ou lorsque toutes les trajectoires auront été épuisées.

Améliorations Le suivi de la trajectoire guide implique une sorte de linéarité dans le déplacement du polygone. Celui-ci se présente toujours d'une façon identique ce qui n'est pas forcément souhaitable. Supposons que le polygone présente une concavité. Rappelons que les collisions se produisent sur les courbures importantes de la trajectoire guide. Une courbure importante implique un obstacle dans la partie interne de la courbure. Si la concavité du polygone se présente vers la partie interne, alors la probabilité de succès du franchissement de la courbure est plus importante. Or le graphe offre la possibilité de réaliser, au niveau des noeuds, des repositionnements (manoeuvres). Il suffit pour cela que le robot s'engage sur un arc secondaire puis reprenne la trajectoire guide initiale (image de la manoeuvre d'une voiture). De plus, ce type de manoeuvre est interéssant en soi au niveau d'un noeud car il minimise la rotation du robot et offre ainsi une meilleure probabilité de succès (voir la figure 14).

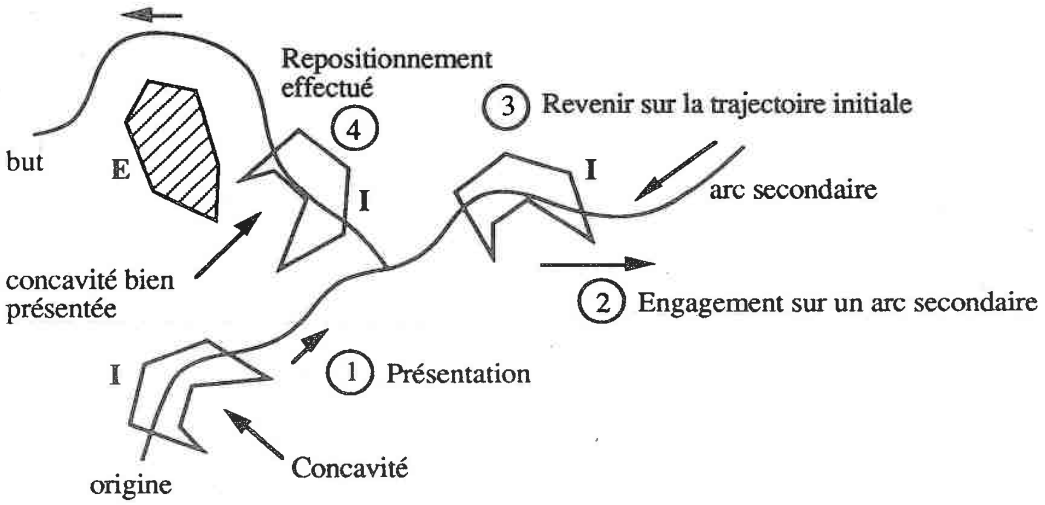


Figure 14 : Repositionnement sur le graphe guide

Il reste bien entendu à détecter une concavité sur un polygone.
Une autre amélioration particulièrement simple consiste à valuer différement les arcs du graphe guide. Les collisions se produisent statistiquement sur les arcs ayant des courbures intenses. Il suffirait d'affecter à ceux ci une valuation plus importante.

Il serait aussi possible de valuer les arcs du graphe avec l'inverse du diamètre du plus grand disque que l'on peut déplacer sans collision sur l'arc. En allant plus loin, il est même possible de supprimer les arcs du graphe qui sont tels que le plus grand cercle déplaçable soit inclus dans le polygone I. On pourrait facilement imaginer d'autres valuations suivant d'autres critères.

Complexité et conclusion La complexité de l'algorithme apparaît à 2 niveaux différents : Au niveau du planificateur global qui recherche des chemins en largeur d'abord puis au niveau du déplacement sur la trajectoire guide proposée.

• Au niveau du planificateur global :
Soit n le nombre d'arcs de l'environnement. On sait que le nombre d'arcs du diagramme de Voronoï est en O(n). Le coût de la recherche en largeur d'abord est en O(n.log(n)) pour produire un plus court chemin. Le nombre de chemins entre 2 noeuds d'un graphe à n arcs est une fonction exponentielle en n mais il est clair qu'en moyenne, si la fonction de valuation des arcs est bien étudiée, alors une trajectoiree sera rapidement trouvée. En effet, il n'est pas possible de prévoir si pour un chemin fourni par le planificateur global, le déplacement va être possible ou non. C'est en le réalisant physiquement qu'on le saura. Si le déplacement échoue, le planificateur global proposera un autre chemin qui sera parcouru. Ce processus est éventuellement répété sur l'ensemble des chemins existants entre les positions de départ et d'arrivée. La complexité, dans le pire cas, du planificateur global est donc exponentielle en n.
• Au niveau du déplacement sur la trajectoire guide :
Soit k le nombre de cellules élémentaires composant la trajectoire. Soit C_Inter(E,I) le coût constant du test d'intersection entre I et E. S'il ne se produit aucune collision, alors le coût du déplacement sur la trajectoire guide est k*C_Inter. Il est proportionnel à la longueur de la trajectoire. En effet, quand T et F sont fixés, le polygone I évolue dans un espace de dimension 1. En cas de collision, les paramètres T et F sont modifiés. On peut supposer que cette modification se fait en moyenne en temps constant donc les collisions n'agravent pas la complexité. Nous concluerons en disant que la complexité est proportionnele à la longueur d'une trajectoire.

Nous avons proposé un ensemble de techniques et de procédures très simples à mettre en oeuvre.pour résoudre le problème du déplacement en translation et rotation d'un polygone dans un couloir quelconque ou parmi un ensemble de polygones. Reprenons les remarques faites à l'issue de la proposition de l'algorithme naïf ainsi que celles concernant l'algorithme exact :

• La trajectoire guide et la technique de positionnement impliquent que le nombre de voisins de la cellule courante dans l'espace des configurations devienne égal à 1, ce qui ne laisse aucun choix. L'exploration ne se fait plus "au hasard" mais est contrainte. Il y a disparition ici de la complexité

exponentielle liée au nombre de cellules de EC. On récupère malheureusement une complexité exponentielle au niveau du planificateur global mais on sait qu'en moyenne, ce n'est pas le cas.

- La constitution même du graphe guide évite des errances dans les concavités des obstacles.
- Les déplacements se tiennent le plus possible éloignés des obstacles.
- La simplicité des algorithmes induit leur robustesse.
- Cette méthode est plus dynamique.que l'algorithme exact. Nous avons en effet proposé un algorithme dynamique pour le diagramme de Voronoï de segments. Nous donnons dans le paragraphe suivant une approche qu'il l'est davantage.

Les figures 15 et 16 donnent des exemples d'exécution.

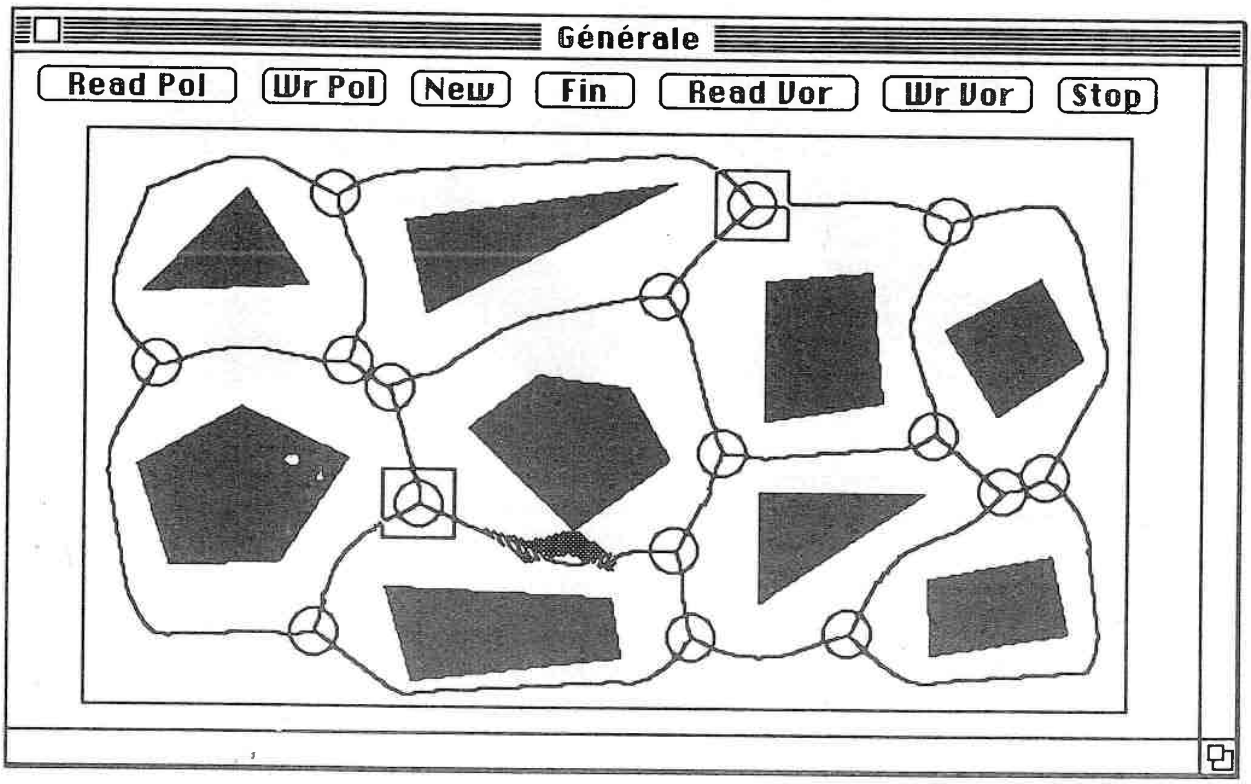


Figure 15 : Déplacement sur le graphe de Voronoï. Les noeuds de départ et d'arrivée sont marqués d'un carré.

4.6 Perspectives

Nous allons proposer ici 2 idées qui nous semblent intéressantes. La première vise à obtenir un algorithme de planification qui soit très dynamique, c'est-à-dire qui puisse prendre facilement en compte de nouveaux obstacles ainsi que leur suppression. La seconde consiste en un mariage entre la méthode exposée dans ce chapitre et celle exposée au chapitre précédent. L'idée est d'employer

la puissance de l'algorithme exact uniquement quand la méthode de résolution des collisions échoue.

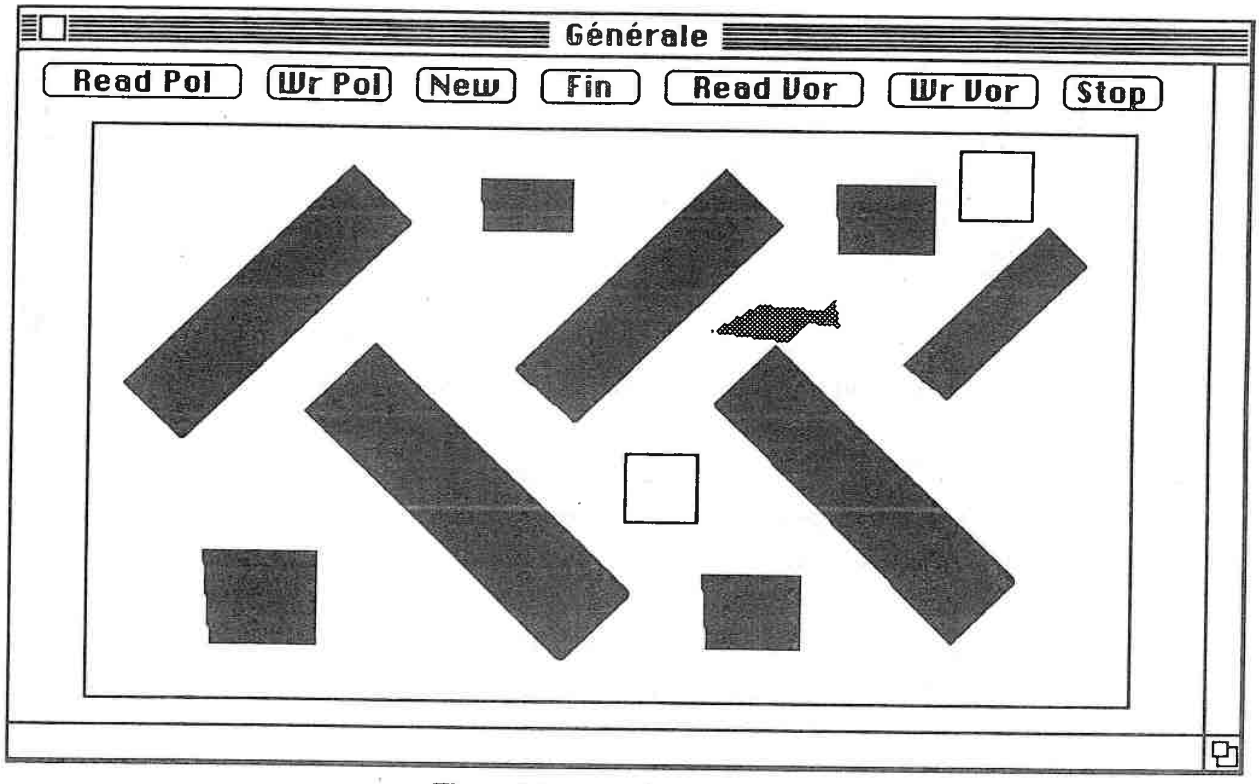


Figure 16 : autre situation

Approche dynamique

L'idée est fort simple : il suffit de déplacer le robot en même temps que l'on crée le diagramme de Voronoï. Les différences avec l'algorithme présenté précédement sont les suivantes :

- Le planificateur global ne peut plus fonctionner car il n'est pas en possession de tout le graphe de Voronoï.
- Une fonction heuristique devra pouvoir décider, au niveau d'un noeud trouvé, de l'arc de Voronoï à explorer (prendre celui qui se rapproche le plus du but). Les arcs en suspens seront empilés.
- En cas d'échec de la procédure de résolution des collisions, il suffit de reprendre au dernier noeud atteint et d'explorer les arcs de Voronoï laissés en suspens.

L'arc de Voronoï en cours de formation est fonction des plus proches obstacles donc l'adjonction d'un obstacle ne modifie pas cette constitution sauf s'il devient le plus proche de tous les obstacles. Dans ce cas, le robot devra être capable de déterminer un nouveau point de Voronoï,

accessible en translation, et continuer l'exploration depuis ce point. Le retrait d'un obstacle se traite de la même façon. La figure 17 à gauche donne un exemple de progression et celle à droite donne un exemple pour la procédure à suivre lors de l'ajoût d'un nouvel obstacle.

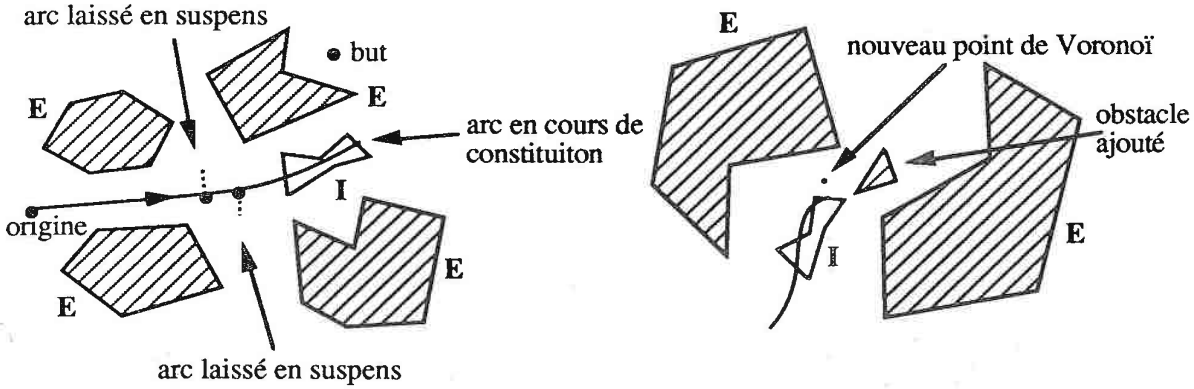


Figure 17 : Déplacement et construction du graphe guide simultanément

Approche mixte trajectoire guide-exacte

Là aussi l'idée est fort simple. Supposons que la procédure de résolution des collisions échoue. Il suffit de faire appel à l'algorithme exact, de lui indiquer un certain but, puis de reprendre l'algorithme par suivi de trajectoire lorsque l'algorithme exact a terminé son travail. Le but peut être obtenu très simplement de la façon suivante : il suffit de continuer le suivi de la trajectoire guide (de façon virtuelle bien sur !) malgré les collisions et de retenir comme but la première position sans collision (voir figure 18).

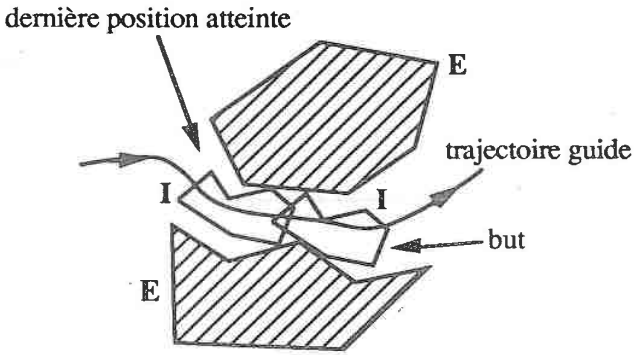


Figure 18 : Méthode mixte : Indication à la méthode exacte d'un sous-but

L'inconvénient majeur de cette solution est bien entendu la nécessité de bâtir le modèle exact. Pour y pallier, il conviendrait de ne bâtir que la portion intéressante du modèle.

Chapitre 5

Algorithmes dans un univers discret et application à la planification de trajectoires

5.1 Introduction

Les objets manipulés dans un univers discret se rapprochent des objets réels, de ceux que nous manipulons chaque jour. Ils ne seront plus connus par une définition analytique (exemple : un polygone définit par la liste de ses sommets) mais par l'ensemble des cellules en faisant partie. Cette grande redondance d'information apporte, nous le pensons, des avantages appréciables. Nous allons montrer que les représentations discrètes des objets peuvent être utilisées pour autre chose que de simples visualisation de ces objets.
Nous avons choisi de nous intéresser aux algorithmes dans un univers discret pour de nombreuses raisons :

• Pour des raisons de robustesse et de stabilité

Les algorithmes classiques de la géometrie algorithmique travaillent généralement sur des réels. Or, les preuves de ces algorithmes ne manipulent que des objets formels (des variables, des constantes exactes etc ...). Les réels n'étant pas correctement pris en compte dans un ordinateur, les preuves formelles ne signifient plus rien. En aucun cas, nous n'avons la preuve que les algorithmes implantés sur tel ou tel type de machine fonctionnent correctement. Donnons ici un exemple d'un problème classique : le calcul de toutes les intersections d'un ensemble de n segments (voir [P & S 85] p 276). Cet algorithme, fort astucieux, est du type plane-sweep (balayage d'un certain ensemble d'évènements). Il prend en entrée la liste triée en x de toutes les extrémités des segments. Supposons que ces segments soient issus d'un calcul et que les extrémités soient représentées par des couples de réels. Si les erreurs de précision sont importantes, alors il se peut que l'ordre fourni par le tri ne corresponde pas à la réalité de l'arrangement des segments. Dans ce cas, l'algorithme de balayage ne prendra pas en entrée un ordre correct et ses résultats seront erronés. Une solution possible est de travailler sur les entiers. Leur représentation dans la machine étant correcte (aux dépassements de capacité près), nous pourrons établir des preuves sur les exécutions des algorithmes.
Nous affirmons d'autre part, que la plupart des algorithmes sont peu stables. Un algorithme est stable si une petite modification dans les entrées n'entraîne pas une grande modification dans les sorties. Donnons un exemple : supposons que nous disposions d'un algorithme de calcul de l'intersection de 2 polygones basé sur la connaissance de toutes les intersections des segments pris

2 à 2. La fonction de base est ici celle calculant l'intersection de 2 segments. Supposons que cette fonction (à tort peut-être) rende "pas intersection" quand 2 segments se touchent par leurs extrémités. Dans l'exemple de la figure 1 à gauche, l'algorithme fonctionnera correctement. Dans celui de la figure 1 à droite, il ne détectera aucune intersection et donnera un résultat faux alors que l'intersection est importante.

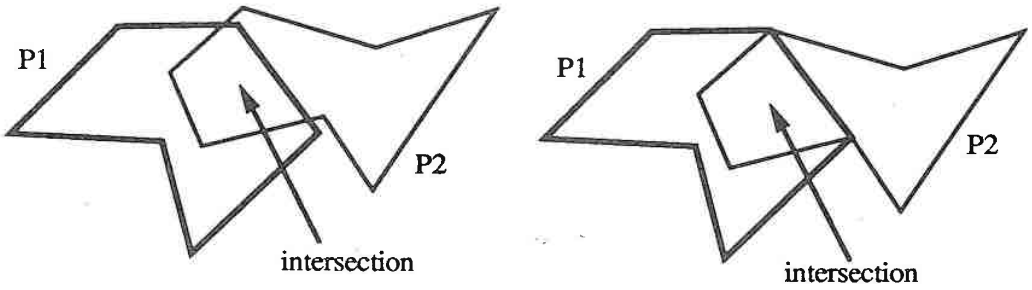


Figure 1 : Stabilité des algorithmes

Il est clair que si les polygones étaient représentés par la donnée exhaustive des cellules composant leur intérieur, alors il ne pourrait se produire de telles situations. D'une façon plus générale, les algorithmes discrets tiennent compte de l'objet dans sa globalité. Manipuler uniquement le bord d'un objet au sein d'un algorithme est un bon moyen de provoquer des erreurs (voir l'exemple ci-dessus).

Donnons un autre exemple qui nous paraît significatif des problèmes rencontrés avec les réels : l'appartenance d'un point à une droite. Si le point est réel (issu d'un calcul) et si les 2 coefficients le sont aussi alors la décision de l'appartenance est un problème délicat. Une solution peut être de décider de l'appartenance si la distance du point à la droite est inférieure à une certaine quantité. Avouons que ce n'est pas satisfaisant. Voir [HOF 90] pour des développements sur cette question.

- **Pour des raisons de simplicité**

Certains algorithmes manipulant des objets discrets deviennent quelques fois très simples. Les manipulations de polygones en sont un exemple. Il est souvent nécessaire de réaliser des opérations ensemblistes sur des polygones. Calculs d'union, d'intersection, de différence ensembliste ou encore de différence entre l'union et l'intersection (Xor). Ces algorithmes ne sont pas triviaux dans la géométrie classique mais ils le deviennent dans la géométrie discrète. Nous verrons plus loin, que les polygones discrets sont représentés par des matrices booléennes. Les opérations citées ci-dessus consistent alors en des calculs booléens (et, ou, xor etc ...) triviaux sur ces matrices. L'appartenance d'un point à un polygone devient elle aussi très simple car il suffit de consulter une certaine matrice aux coordonnées du point.

• Pour des raisons de localité et d'accés direct

Les algorithmes discrets, de part la grande redondance d'information qu'ils manipulent, vont permettre de traduire une idée importante de l'algorithmique géométrique : la localité. Un inconvénient majeur de beaucoup d'algorithmes classiques est la non prise en compte de la notion de localité. Ils manipulent souvent des objets qui de part leur "éloignement" (au sens large) d'une certaine zone considérée, n'ont à priori aucun rôle à jouer dans la résolution locale. Le calcul du plus proche objet parmi n, d'un certain point donné en est un exemple. L'algorithme naïf manipulera tous les objets et calculera un minimum même si certains des objets sont très éloignés. Une solution, dans un univers composé de cellules élémentaires, est d'examiner circulairement les cellules depuis le point considéré et de retenir le ou les objets correspondant à la première cellule "non vide" trouvée. Une autre conséquence de la redondance d'information est la possibilité d'accéder en direct à l'information et de proposer ainsi des réponses en temps constant. Si les polygones d'une certaine scène sont représentés par des ensembles de cellules à "1" alors l'appartenance d'un point à un polygone quelconque de la scène se décide en temps constant par simple examen du point considéré.

• Pour des raisons de parallélisation

Les manipulations de matrices booléennes sont aisément parallélisables puisqu'il s'agit de faire des opérations logiques sur des cellules prises 2 à 2. En particulier, ces manipulations s'accorderaient d'un parallèlisme de type "beaucoup de processeurs peu puissants". Les architectures systoliques seraient ici encore d'un grand secours. Il existe déja des algorithmes de calcul de diagrammes de Voronoï sur de telles architectures.(Voir [DEH 89]).
Nous avons proposé au chapitre 3, l'exemple de la parallélisation du diagramme de Voronoï.

• Pour des raisons de facilité de conception d'architectures spécialisées

Les opérations de base sont des opérations élémentaires qui devraient facilement pouvoir être câblées. La simplicité des algorithmes est elle aussi un gage de transformations aisées en architectures spécialisées.

• Pour les performances dans certains cas

Les algorithmes manipulant des objets discrets peuvent donner, dans l'absolu, d'excellents résultats. Le remplissage de l'intérieur d'un polygone en est un exemple.

• Inconvénients

2 inconvénients apparaissent : le premier est le coût mémoire important. Nous verrons plus loin, qu'il est néanmoins possible de créer une représentation compacte des objets ne pénalisant pas trop les algorithmes. De plus, les ordinateurs modernes possèdent des mémoires de plus en plus importante et de moins en moins coûteuses. Le second inconvénient, plus grave, est la perte de la topologie des objets manipulés et des résultats obtenus. Le calcul de l'intersection est trivial, mais le résultat obtenu est un certain ensemble de cellules sans topologie. Il convient, à l'issue d'un traitement, connaissant le type des objets obtenus, d'extraire des définitions analytiques de la masse brute d'information.

5.2 Les objets discrets

Les objets manipulés seront des objets plan ou à 3 dimensions. Nous introduisons donc une discrétisation de $\mathbb{R}^2$ ou $\mathbb{R}^3$ (se rapporter à §4.1 pour la définition d'une discrétisation). Pour notre étude, l'environnement ainsi que le système robotique I seront des objets à 2 dimensions. Nous introduisons donc une discrétisation du plan réel en cellules carrées d'arête h. Ces cellules seront indicées par des éléments de $\mathbb{Z}^2$.
Il n'existe qu'un type d'objet dans l'espace discret. Il est défini par la donnée de toutes les cellules appartenant à l'objet. On dira qu'une cellule (i,j) appartient à l'objet o si elle est intersectée par cet objet.
La donnée de toutes les cellules peut se faire de plusieurs manières :

• Par la donnée d'un système d'équations ou d'inéquations. La doite discrète (voir [REV 90]) est définie par l'ensemble des couples (x,y) de $\mathbb{Z}^2$ tels que $c \leq a.x + b.y \leq c'$ où a,b,c,c' sont réels (voir figure 2 à droite). Ici, c et c' fixent "l'épaisseur" de la droite discrète.

• Par la donnée pure et simple de toutes les cellules appartenant à l'objet.

Plus généralement, nous emploierons le terme de région pour parler des objets. Une région sera un certain sous-ensemble de $\mathbb{Z}^2$. Ce concept de région, vu sa pauvreté topologique, offre une souplesse totale. Une région peut être ainsi non connectée et présenter des trous. Les "formes" les plus variées peuvent être prise en compte. Remarquons encore une fois que ce concept tient compte non seulement de l'idée de bord de l'objet mais aussi de celle d'intérieur de l'objet. Enfin, nous avons étendu la notion de région par la possibilité d'affecter à chaque cellule d'un objet un numéro (un entier). Cette numérotation, permet entre autre, de différencier les régions ou de réaliser des traitements particuliers sur celles-ci. La recherche des composantes connexes d'une région en est un exemple. Nous considèrerons arbitrairement qu'une cellule n'appartenant à aucun objet sera numerotée par zéro. D'un point de vue pratique et pour fixer les idées, disons ici que

l'environnement ainsi que le système robotique seront représentés dans un tableau à 2 dimensions d'entiers. Les cases affectées d'un entier strictement positif seront considérées comme appartenant aux objets de la scène. Les figures 2 et 3 donnent des exemples d'objets discrets et de régions.

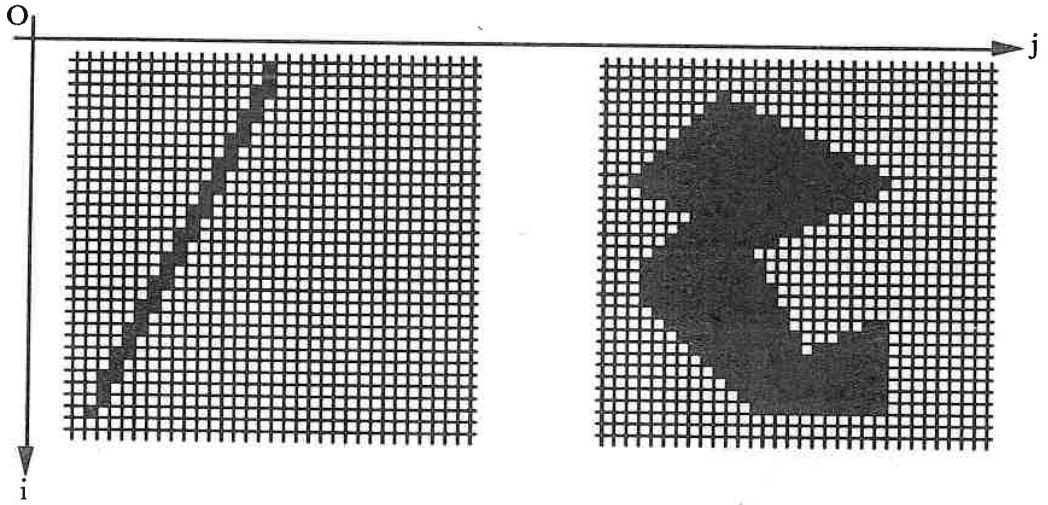


Figure 2 : Droite discrète et polygone discret

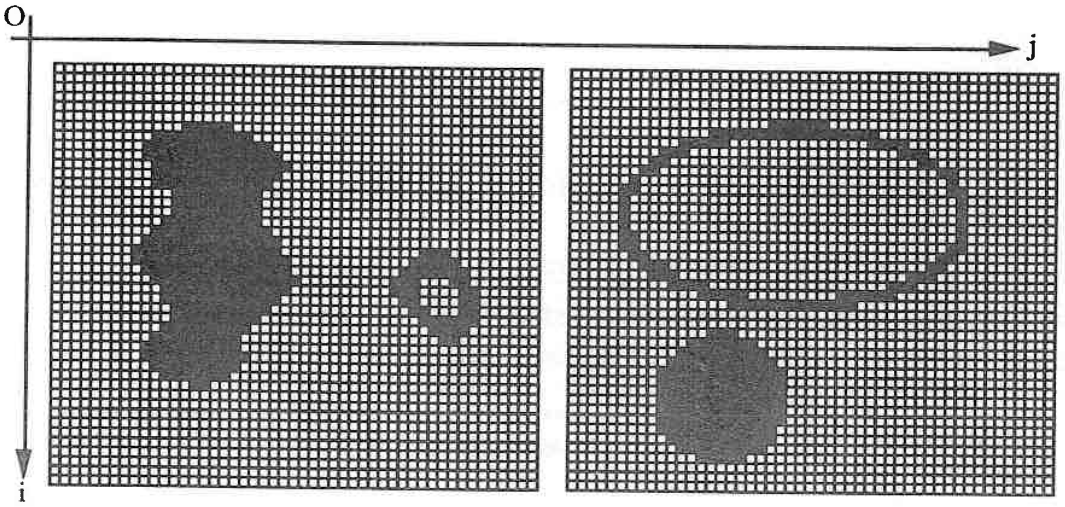


Figure 3 : Exemples de régions

Conclusion Dans cette étude, nous manipulerons des objets sous la forme de matrices binaires ou dont les éléménts sont des entiers lorsque nous désirons différencier les composantes d'une région. De plus, nous adjoignons à chaque région les dimensions du rectangle englobant minimal parallèle aux axes i et j. Cette information suplémentaire permet d'optimiser en moyenne de nombreuses opérations sur les régions. La figure 4 donne l'exemple d'un objet représenté par une matrice binaire puis du même objet quand une numérotation a été introduite. Notons que cette

numérotation ne sert pas uniquement à différencier les composantes connexes. Enfin, précisons que l'environnement ainsi que le système robotique, sont inclus dans une fraction rectangulaire du plan discret et que les positions des différents objets sont connues dans un repère unique (i,O,j).

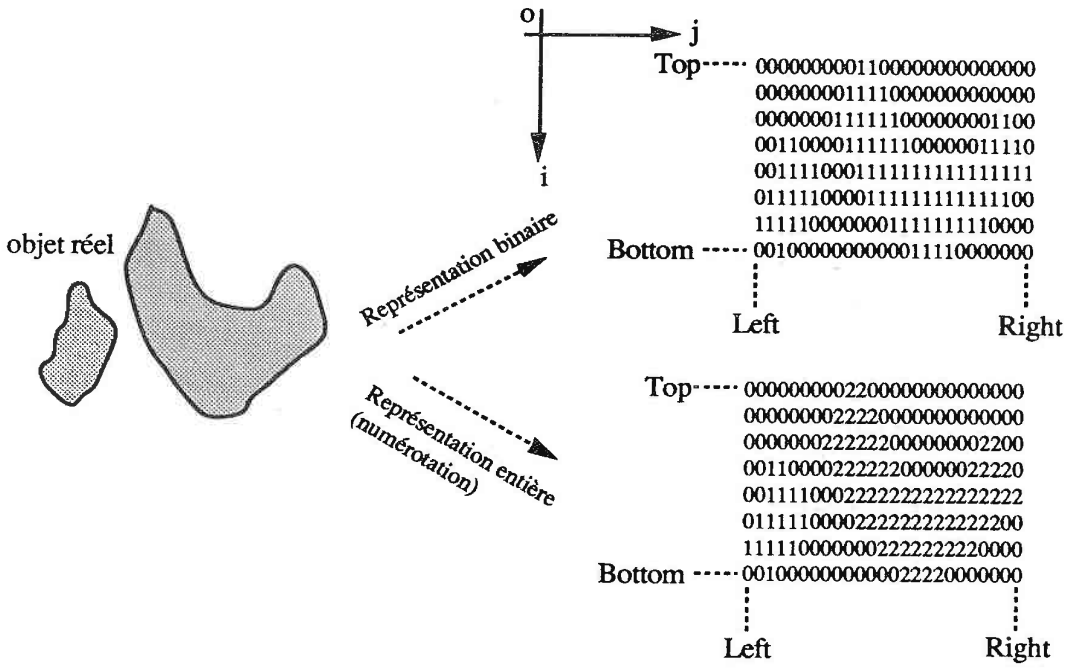


Figure 4 : Représentation binaire et entière des objets

5.3 Liens entre objets discrets et objets analytiques

Soit o un objet analytique (exemple : une droite d'équation connue, un polygone connu par la liste de ses sommets). Comment obtenir l'objet discret correspondant ? Symétriquement, étant donné un objet discret (dont on connait tout de même le type) comment obtenir sa définition analytique ?
Le passage d'un univers à un autre sera applelé projection.

5.3.1 Projection d'objets analytiques dans le plan discret

• Projection d'un segment de droite

Etant donné 2 points A et B de coordonnées (A_x,A_y) et (B_x,B_y), trouver l'ensemble des cellules discrètes appartenant au segment [A,B]. Les points A et B correspondent aux 2 cellules C_A et C_B. L'algorithme réalisant cette projection est celui bien connu de Brésenham ou Lucas (voir [L &R 88]). Cet algorithme très astucieux ne manipule que des entiers et n'utilise pas l'équation de la droite proprement dite. Etant donné la dernière cellule trouvée (x_{i-1},y_{i-1}), la suivante (en supposant

que le segment se trouve dans le premier octant) sera soit en $(x_{i-1} + 1, y_{i-1})$ soit en $(x_{i-1} + 1, y_{i-1}+1)$. Ce choix est levé par une relation de récurrence simple. Le segment discret obtenu est une suite connexe de cellules. Il resterait à écrire un algorithme permettant l'obtention de segments d'épaisseur quelconque.

• Projection d'un polygone

L'algorithme de projection d'un polygone se déduit aisément du précédent. Il suffit de projeter chacun des segments et de "noircir" les cellules appartenant au contour obtenu.

• Projection d'un cercle

Cet algorithme est une généralisation de celui de Bresenham. iL est dû à Michener (voir [L &R 88]). Il manipule uniquement des entiers alors que l'équation d'un cercle induit l'utilisation de réels. Seules les opérations d'addition, de soustraction et de multiplication par 4 sont nécessaires. Le tracé obtenu est aussi d'épaisseur 1. On peut déduire de cet algorithme celui projetant un disque. Notons enfin, qu'il est possible de projeter de la même manière des courbes Splines, de Bézier etc...

5.3.2 Projection d'objets discrets vers des objets analytiques

L'opération inverse de celle définie précédement est quelques fois indispensable. Les techniques employées ici sont bien connues des personnes travaillant en traitement d'image. Il s'agit de techniques de segmentation. Le problème est délicat car ambigu. Etant donné un objet discret, il peut lui correspondre plusieurs objets analytiques possible. Un segment peut être confondu avec une portion de courbe etc... Pour notre part, les objets discrets obtenus seront toujours des polygones. Les techniques de segmentation donnent ici de bons résultats. Elles fonctionnent de la façon suivante : la première phase consiste à réaliser une extraction de contours. Le résultat est une liste chaînée de points appartenant à la région. La seconde phase consiste à réaliser une approximation polygonale sur la liste chainée de sommets. Etant donné 2 sommets s1 et s2, l'algorithme calcule l'aire algébrique définie par la droite (s1,s2) et le contour défini par la liste des sommets entre s1 et s2. Si cette aire algébrique dépasse un certain seuil alors on divise récursivement le problème. L'algorithme choisi un sommet s3 intermédiaire entre s1 et s2 et résoud le problème avec (s1,s3) puis avec (s3,s2). Ce processus est itéré tant que les aires algébriques trouvées dépassent un certain seuil (voir [TON 87]).

5.4 Opérations élémentaires sur les objets discrets

Nous allons définir ici quelques opérations de base sur les objets discrets et montrer ainsi leur simplicité ainsi que la sécurité importante des algorithmes mis en jeu.

• Compactage d'une région

Nous allons indiquer un algorithme qui prend en entrée l'ensemble des cellules appartenant à une certaine région et rend en résultat une définition compacte. Il existera une bijection entre les régions et leurs définitions compactes. L'idée du compactage est la suivante : Etant donné la ligne i-1 de la région, quelles différences existe-t-il entre cette ligne et la ligne i ? On comprend aisément que si la région est compacte alors le gain sera important. Il l'est pour l'exemple d'un polygone. Les différences entre la ligne i-1 de la région et la ligne i peuvent s'exprimer par l'ensemble des intervalles suivant l'axe des j qu'il est nécessaire de complémenter pour obtenir la ligne i. L'information relative à la ligne i aura pour forme :

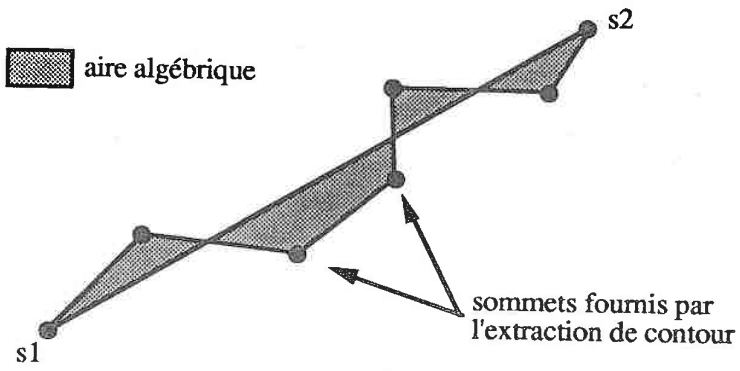


Figure 5 : Approximation polygonale d'une suite de sommets

$i,deb_1,fin_1,deb_2,fin_2,...,deb_k,fin_k$ où i est le numéro de ligne et où deb_i,fin_i signifie qu'il est nécessaire de complémenter les cellules de la ligne i-1 entre les indices deb_i et fin_i suivant l'axe des j. Si la ligne i est identique à la ligne i-1 alors son information n'existe pas. Enfin, nous ajouterons à cette définition compacte les dimensions du rectangle englobant minimum parallèle aux axes. Ceci permet d'optimiser en moyenne certaines opérations entre les régions, telles les intersections. La figure 6 donne un exemple de définition compacte.
Ce type de représentation est utilisé sur certains ordinateurs à vocation graphique.
Pour la suite, nous considèrerons que nous possèdons la représentation matricielle des objets. Le point (i,j) d'une certaine région R vaudra R(i,j) et le rectangle englobant la région R vaudra Rect(R). L'élément neutre sur les régions sera appelé Région_Vide.

• **Appartenance d'un point à une région**

Cette fonction prend en entrée la définition matricielle d'une région et un point (i,j) et rend vrai si le point appartient à la région.

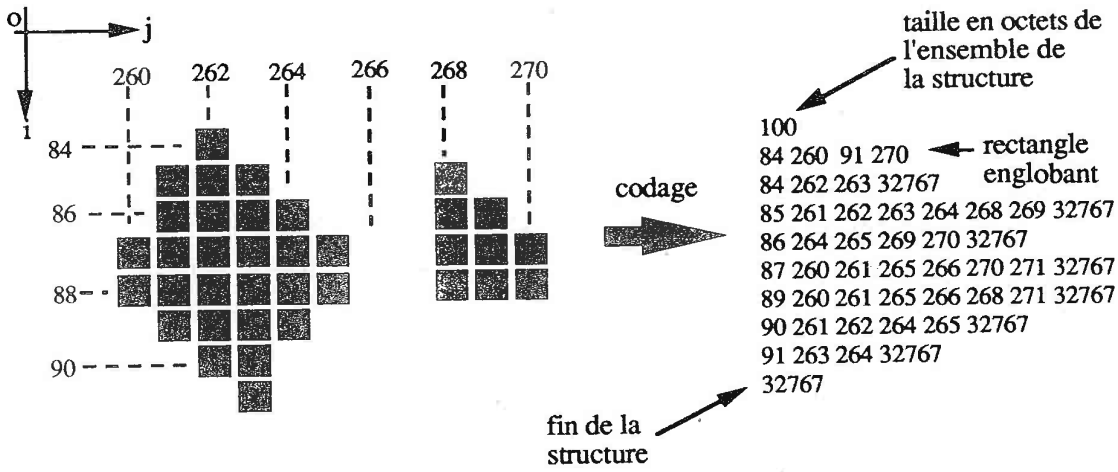


Figure 6 : Région discrète et sa définition compacte

Appartenance(R,(i,j))

Début

Si R(i,j) ≠ 0 alors retourner(vrai) sinon retourner(faux).

Fin

Cette opération s'exécute en temps constant.

• **Intersection de 2 régions**

Cette opération est d'un grand intérêt. Elle est utile en particulier pour détecter des collisions entre objets ou pour construire des espaces libres.

Intersection(R_1,R_2)

début

Si Rect(R_1) ∩ Rect(R_2) = ∅ alors retourner(Région_Vide)

Soient L,R,T,B les coordonnées du rectangle intersection entre Rect(R_1) et Rect(R_2)

Résultat ← matrice_vide

(to,bo,le,ri) è (+∞,-∞,+∞,-∞) / initialisation du rectangle englobant de la région résultat */*

Pour i de T à B, Pour j de L à R répéter

Résultat(i,j) ← R_1(i,j) et R_2(i,j)

```
Si Résultat(i,j) ≠ 0 alors    si i < to alors to = i        /* gestion du rectangle    */
                              si i > bo alors bo = i        /* englobant le résultat   */
                              si j < le alors le = j
                              si j > ri alors ri = j
fin pour
Rect(Résultat) ← (to,bo,le,ri)
Retourner(Résultat)
fin
```

L'opérateur "et" est un opérateur logique. Si $R_1(i,j)$ ou $R_2(i,j)$ sont des numéros au lieu d'être des booléens alors on considére qu'ils valent 1. La complexité de cet algorithme est proportionnele à la taille du rectangle intersection. Remarquons la grande simplicité de l'algorithme et sa totale sécurité.

Il faut bien prendre conscience ici, que cet algorithme est local. Si R1 modélise le robot et si R2 modélise l'environnement (on peut supposer que le rectangle englobant R1 est inclus dans le rectangle englobant R2) alors les 2 itérations i et j ont lieu dans le rectangle R1 et donc seule l'information de l'environnement réduite à ce rectangle est manipulée. Toute partie de l'environnement n'appartenant pas à ce rectangle, n'est pas manipulée. La redondance d'information, nous permet un accés direct et donne ainsi une vue locale de l'univers.

• Union, différence ensembliste, Xor, etc... de 2 régions

Ces opérations de base fonctionnent sur le même principe que pour l'intersection. Il suffit de modifier l'opérateur logique au centre de l'algorithme. D'une façon générale, la complexité de ces algorithmes est proportionnelle à la taille de la sortie.

• Recherche des composantes connexes d'une région

Cette décomposition est essentielle. Elle permet d'ajouter de l'information à une région binaire. L'algorithme procède en balayant la région de haut en bas, en numérotant les cellules à l'aide d'une règle simple et en maintenant une table de classes d'équivalence sur les numéros. Cette table est modifiée par les relations de connexité découvertes par le balayage. La phase finale consiste à rebalayer les cellules de la région et à les renuméroter à l'aide du représentant unique de la classe à laquelle appartient le numéro de la cellule examinée. Nous considérons des relations de 8-connexité.

Fonction de service

La fonction Terminal(i,j) rend le numéro représentant la classe d'équivalence à laquelle appartient R(i,j). Elle effectue un chaînage dans la table. Si celle-ci contient les relations d'équivalence suivantes : 3→7, 7→13, 13→14, 14→14 alors si R(i,j) contient 3, Terminal(i,j) rend 14. Ici, le représentant de la classe à laquelle appartiennent les numéros 3,7,13 et 14 est 14.

Nous allons décrire ici précisément le fonctionnement de la règle qui affecte un numéro à la cellule courante de R. La fonction c prend en entrée une certaine cellule et modife les structures de données en jeu suivant les cas.

Règle(C)

début

Si C a un voisin à sa gauche

alors C prend le même numéro que lui

Pour chaque voisin (les 3 du dessus) au-dessus de C Répéter

Si R(C) ≠ R(voisin)

alors affecter au terminal de voisin le terminal de C

fsi

fpour

sinon Si C a un ou des voisins parmi les 3 cellules du dessus

alors il prend le même numéro que celle la plus à gauche parmi les 3

Pour chaque voisin du dessus autre que le plus à gauche Répéter

Si R(C) ≠ R(voisin)

alors affecter au terminal de voisin le terminal de C

fsi

fpour

sinon affecter à C un nouveau numero

dire dans la classe d'équivalences que ce numéro est équivalent à ce numéro.

fsi

fsi

fin

La figure 7 donne un exemple d'application de cette règle sur les cellules d'une région.

Il ne subsiste plus que 2 classes d'équivalence dans la table. Celles de représentant 3 et 4, donc il y a 2 composantes connexes dans cette région. Un recadrage, à partir de 1, des numéros restants et un balayage sur les cellules de la région donneront le résultat voulu.

La complexité de cet algorithme est proportionnelle au nombre de cellules de la région. Pour chacune d'elle, un parcours chaîné est éventuellement nécessaire pour obtenir le représentant de la

classe courante. Ce parcours est au plus de longueur le nombre de cellules de la région, mais cette complexité dans le pire cas ne signifie rien. La complexité globale est donc proportionnelle au nombre de cellules de la région multiplié par le coût moyen de recherche d'un représentant dans la table.

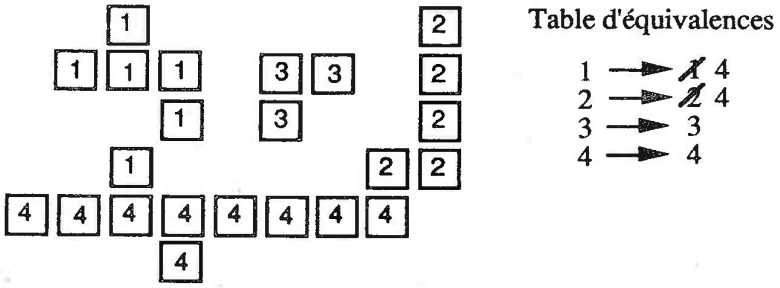


Figure 7 : Recherche des composantes connexes d'une région

• Recherche d'une trajectoire discrète au sein d'une région connectée

Cette opération est elle aussi du plus grand intérêt. En géométrie classique, elle a donné lieu à de nombreux travaux (voir [CKV 87], [LEE 83], [LP & W 79]). Elle consiste à rechercher une suite connexe de segments au sein d'un certain environnement en évitant les obstacles. Le problème est identique si on recherche un chemin à l'intérieur d'un polygone. Le problème peut consister simplement à recherche un chemin ou bien un chemin optimum au sens d'un certain critère. Pour notre part, il s'agira simplement de la recherche d'un chemin.
Rappelons qu'une trajectoire discrète (ou chemin) est une suite connexe de cellules. Les extrémités de la trajectoire entre lesquelles la trajectoire est recherchée seront des données du problème.

Définition On dira qu'une CCL est une suite connexe de cellules sur une même ligne (c'est à dire des cellules de même ordonnée i). Une ligne d'une région peut possèder plusieurs CCL. Celles-ci seront séparées par au moins une cellule vide. Une CCL sera notée r.s où r est le numéro de ligne et où s est l'ordre de la CCL dans la liste des CCL de la ligne considérée suivant les j croissants. Dans l'exemple de la figure 8, à la ligne 3, les 2 CCL sont numérotées 3.1 et 3.2.

L'algorithme bâtit un graphe de connexité sur l'ensemble des CCL. Les CCL seront les noeuds du graphe. 2 CCL i.j et r.s seront reliées dans le graphe si et seulement si il existe une cellule (a1,b1) de i.j voisine d'une cellule (a2,b2) de r.s.
La recherche du chemin sera alors équivalente à la recherche d'un chemin dans le graphe de connexité des CCL. Soient i.j et r.s 2 CCL entre lesquelles il existe un arc dans le graphe. Soit [I1,J1] l'intervalle suivant les j tel que $\forall$ j $\in$ [I1,J1], les cellules (i,j) et (r,j) sont non vides. La communication entre les CCL i.j et r.s aura lieu au niveau de la cellule située au milieu de l'intervalle [I1,J1]. Ceci aura une signification robotique. Les déplacement générés seront en

moyenne, le plus possible éloignés des obstacles. La figure 8 donne un exemple des points de communication entre les CCL. La figure 9 donne un exemple du graphe abstrait issu des relations de connexité entre les CCL. La figure 10 donne un exemple du chemin obtenu entre 2 positions données. La recherche d'un chemin sera réalisée au moyen de l'algorithme A*. Le critère discriminant sera la longueur euclidienne d'un chemin, c'est-à-dire la somme des translations élémentaires qu'il contient.

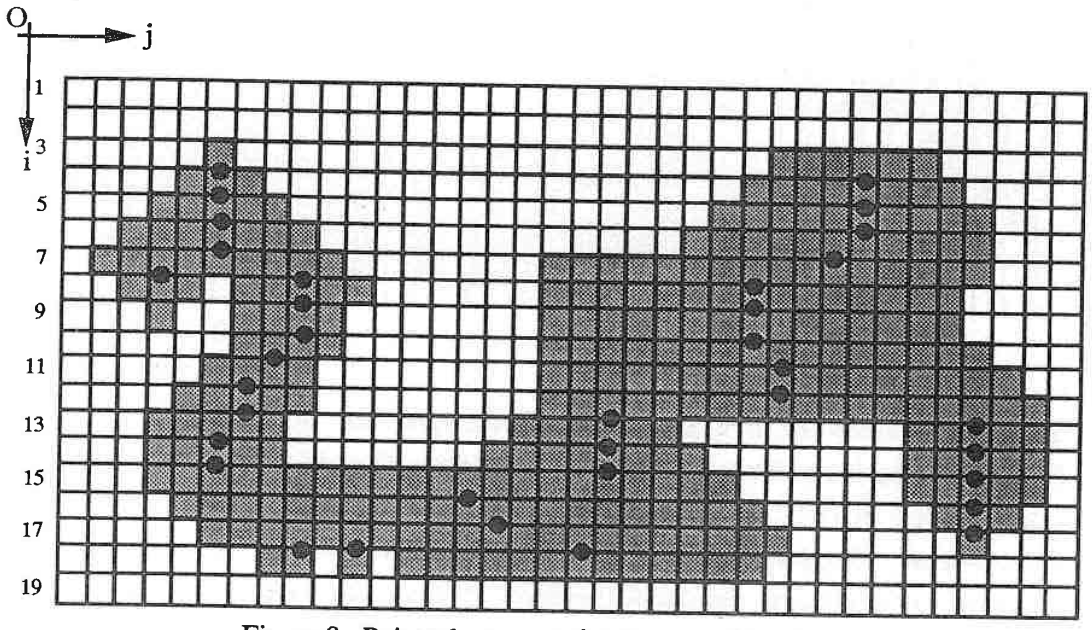


Figure 8 : Points de communication entre les CCL

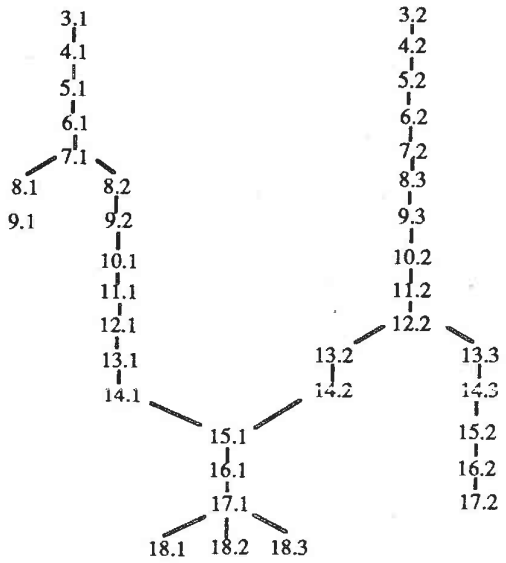


Figure 9 : Graphe de connexité entre les CCL

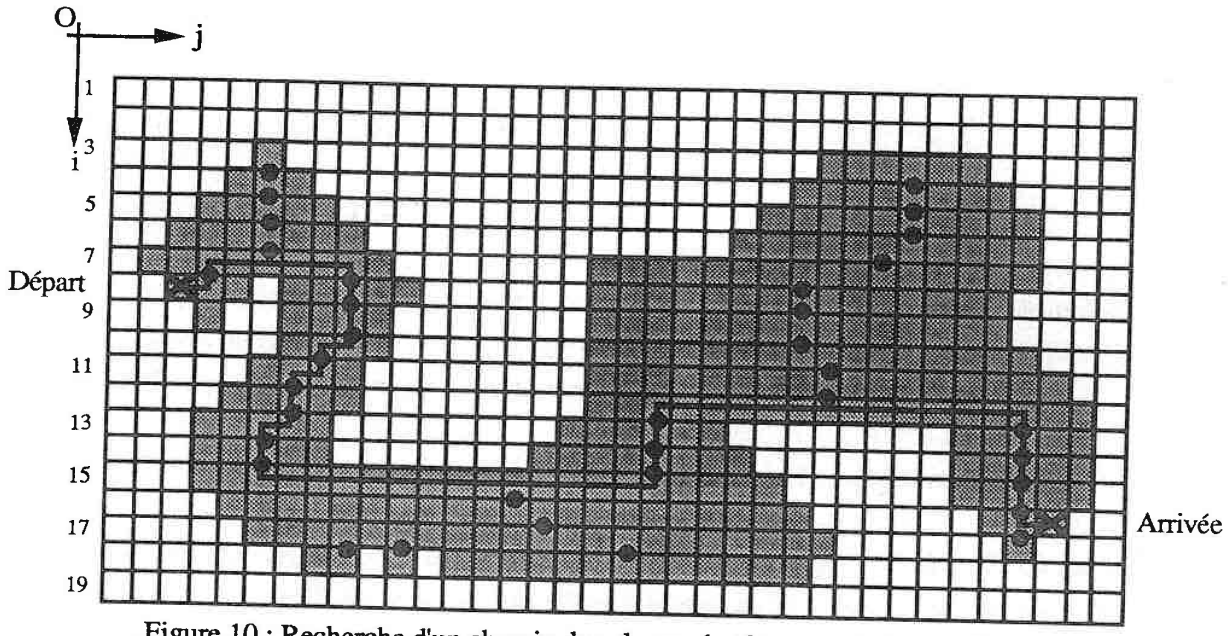


Figure 10 : Recherche d'un chemin dans le graphe de connexité des CCL

Complexité et conclusion La complexité de l'algorithme A* de recherche en largeur d'abord est égale au min entre N^2 et M.log(N) où N est le nombre de sommets du graphe (ici les CCL) et M le nombre d'arcs (voir [TOU 88] p 157). En moyenne, vu le type d'objets que nous manipulons (des polygones discrets), le nombre d'arcs est égal au nombre de sommets et le nombre de sommets est proportionnel à la "hauteur" (exprimée en cellules) de la région. La complexité est donc de l'ordre de N.log(N) en moyenne pour le type de situation qui nous intéresse.

5.5 Application au déplacement sans collision dans un univers connu

Nous proposons ici un algorithme manipulant uniquement des objets discrets et qui résoud le problème du déplacement en translation et rotation d'un polygone quelconque au sein d'un environnement composé de polygones quelconques. L'univers est supposé connu à l'avance. Il s'agit du même problème que celui traité au chapitre 2 mais la solution proposée ici est complètement différente. Le but n'est pas de connaître la topologie exacte de l'espace libre mais de bâtir un modèle simple suffisamment précis pour autoriser des déplacements. Ceux-ci, à l'opposé de ceux proposés au chapitre 2, ne se feront plus en contact avec l'environnement ce qui constitue une amélioration importante. Les idées concernant la discrétisation interviennent ici à 2 niveaux : l'espace libre sera découpé en un certain nombre entier de tranches et les opérations réalisées sur les tranches seront elles mêmes discrètes. Cet algorithme est exact à la double discrètisation près. Des moyens de calculs importants et une mémoire suffisante peuvent permettre une discrétisation fine et obtenir ainsi un algorithme se rapprochant d'un algorithme exact.

Nous reprenons les définitions introduites au chapitre 2. Le système robotique I est composé de m arcs et l'environnement de n arcs. Les polygones obstacles sont ceints dans un polygone englobant. L'espace libre est ainsi borné.

5.5.1 Calcul de P(I,E,α)

Nous nous plaçons désormais dans l'univers discret. Rappelons que P(I,E,α) est l'ensemble des cellules (i,j) telles que $P_{(i,j)}(I) \cap E = \varnothing$. ($P_{(i,j)}(I)$ est le positionnement de I suivant la cellule (i,j), c'estàdire la superposition du sommet de référence I_o de I avec le centre de la cellule (i,j)).

Définition Soit $P_{(i,j,I)}(i_j)$ le positionnement de l'arc i_j de I quand I est positionné en (i,j).

Définition Soit $Q(i_j,e_k)$ l'ensemble des cellules (i,j) telles que $P_{(i,j,I)}(i_j) \cap e_k \neq \varnothing$. $Q(i_j,e_k)$ est un parallèlogramme dont 2 des arcs sont parallèles à i_j et dont les 2 autres sont parallèles à e_k. Dans l'exemple de la figure 11 à gauche, $I_0 \notin Q(i_j,e_k)$ donc il n'y a pas intersection entre i_j et e_k, par contre à droite, $I_0 \in Q(i_j,e_k)$ donc il y a intersection entre i_j et e_k.

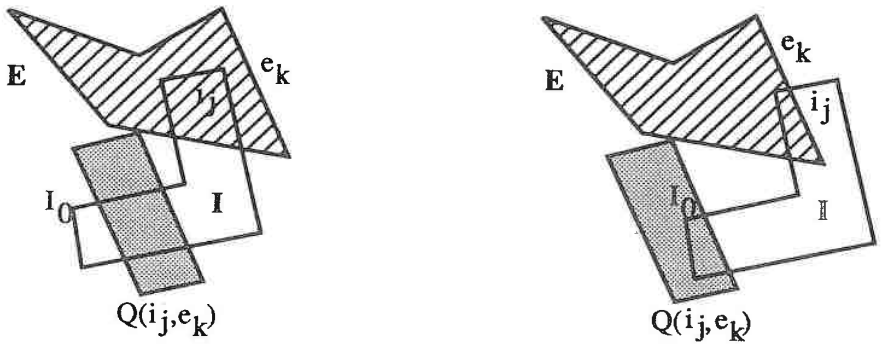


Figure 11 : $Q(i_j,e_k)$

Définition Soit Ω l'espace entre les obstacles (à ne pas confondre avec l'espace libre). Les trajectoires du système robotique ont lieu dans Ω . Ω est de dimension 2.

Nous pouvons désormais donner une définition constructive de l'espace libre P(I,E,α) :

$$P(I,E,\alpha) = \Omega - \bigcup_{i_j \in I,\, e_k \in E} Q(i_j, e_k)$$

Cette définition donne exactement l'algorithme à suivre pour obtenir P(I,E,α). Il suffit de réaliser une suite de différences ensemblistes entre Ω et tous les parallèlogrammes formés par les confrontations entre les arcs de E et ceux de I. Une différence ensembliste, dans l'univers discret, revient à une opération simple entre des matrices binaires. L'algorithme s'effectue en 2 phases :

• La première phase consiste à obtenir Ω . Il s'agit de différences ensemblistes entre le polygone englobant et les autres polygones de E. Rappelons que ces polygones discrets sont d'abord le résultat d'une projection des polygones analytiques.

• La seconde phase consiste à retirer à Ω la suite des parallélogrammes.

Nous savons que $P(I,E,\alpha)$ est formé d'un ensemble de polygones dont les sommets sont des double-contacts. Il ne sera pas nécessaire ici de réaliser une projection de l'objet discret obtenu dans l'ensemble des polygones. Néanmoins il sera nécessaire de réaliser une décomposition en composantes connexes. Notons, que nous sommes alors capables de répondre en temps constant à la question :"existe-t-il un déplacement en translation pure pour I entre les configurations (x_{dep},y_{dep}) et (x_{fin},y_{fin}) ? " puisqu'il suffit de vérifier que les cellules (x_{dep},y_{dep}) et (x_{fin},y_{fin}) de $P(I,E,\alpha)$ sont affectées d'un même numéro. Si c'est le cas, un chemin entre ces positions extrêmes est obtenu en utilisant l'algorithme de recherche d'un chemin dans une région connectée. Ce chemin sera en moyenne éloigné des obstacles.

5.5.2 Calcul de P(I,E)

L'idée vient de la définition même de P(I,E). Nous avions établi au chapitre 2 le résultat suivant :

$P(I,E) = \bigcup_{\alpha} P(I,E,\alpha)$, $\alpha \in [0,2\pi[$

L'union dont il est question ici est une union continue. Nous allons, pour notre part, discrétiser cette union en une suite finie de "tranches" de l'espace libre P(I,E) et réaliser des liens entre les différentes composantes connexes de $P(I,E,\alpha)$.

Résultat Nous introduisons une numérotation sur les composantes connexes de $P(I,E,\alpha)$ (numérotation issue de l'algorithme de recherche des composantes connexes). Notons $C_{i,\alpha}$ la composante de numéro i de $P(I,E,\alpha)$. Rappelons que $C_{i,\alpha}$ est un polygone. Soient $C_{i,\alpha}$ et $C_{j,\alpha+d\alpha}$ 2 composantes telles que $C_{i,\alpha} \cap C_{j,\alpha+d\alpha} \neq \varnothing$. Soit (nx,ny) une cellule de cette intersection. Nous avons par définition $P_{(nx,ny,\alpha)}(I) \cap E = \varnothing$ et $P_{(nx,ny,a+d\alpha)}(I) \cap E = \varnothing$. Ces 2 positionnements sont libres. Nous supposerons, que si $d\alpha$ est suffisament petit alors il existe une rotation libre de I, de centre son point de référence, entre les angles α et $\alpha + d\alpha$. Ce résultat provient de la continuité du modèle exact (voir figure 12).

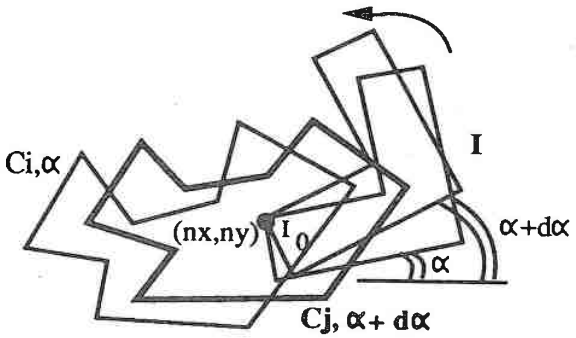


Figure 12 : Rotation élémentaire

Définition On dira que le point (nx,ny) est un point de communication entre les composantes Ci,a et Cj,a+da. Les rotations effectuées par le système robotique I se feront via des points de communication correctement choisis.

Remarque Le choix de dα est difficile. Il a été essentiellement guidé par l'expérience. Nous avons retenu, au niveau du logiciel implanté, une valeur de 2 degrés.

Après avoir introduit la notion de point de communication, nous pouvons désormais définir le schéma élémentaire de déplacement au sein du modèle final. Ce schéma élémentaire sera composé de translations entre des points de communication d'un certain P(I,E,α) et de rotations d'angle ±dα au niveau des points de communication.

Schéma élémentaire de déplacement

Soient $C_{i,\alpha-d\alpha}$, $C_{j,\alpha}$ et $C_{k,\alpha+d\alpha}$ 3 composantes connexes appartenant respectivement à P(I,E,α-dα), P(I,E,α) et P(I,E,α+dα) et telles que $C_{i,\alpha-d\alpha} \cap C_{j,\alpha} \neq \varnothing$ et $C_{j,\alpha} \cap C_{k,\alpha+d\alpha} \neq \varnothing$. Soit (nx,ny) un point de communication entre $C_{i,\alpha-d\alpha}$ et $C_{j,\alpha}$ et soit (mx,my) un point de communication entre $C_{j,\alpha}$ et .$C_{k,\alpha+d\alpha}$. En considérant les α croissants, on dira que (nx,ny) est un point de communication d'entrée pour la composante $C_{j,\alpha}$ et que (mx,my) en est un de sortie pour cette même composante. Supposons qu'il existe une trajectoire discrète dans la composante $C_{j,\alpha}$ entre les cellules (nx,ny) et (mx,my). Le schéma élémentaire de déplacement est alors le suivant : supposons que le système robotique I soit positionné en (nx,ny) avec une rotation égale à α-dα. I effectue une rotation, dans le sens positif, de dα. Il effectue ensuite une translation le long de la trajectoire discrète entre (nx,ny) et (mx,my). Il effectue alors une rotation de dα. Il peut alors se déplacer en translation dans la composante $C_{k,\alpha+d\alpha}$.
D'une façon plus générale, une intersection entre 2 composantes peut être non connexe. Un point de communication existera alors pour chaque composante des intersections. Il devra exister, pour

une certaine composante de P(I,E,α), des trajectoires dicrètes entre tous les points de communication d'entrée et tous les points de communication de sortie.
La figure 13 résume ces notions. Les points de communication d'entrée dans les composantes de P(I,E,α-dα) seront numérotés e_{i1},e_{i2},... et ceux de sortie avec s_{i1},s_{i2},... Idem pour ceux de P(I,E,α) avec e_{j1},e_{j2},... et s_{j1},s_{j2},... et pour ceux de P(I,E,α+dα) avec e_{k1},e_{k2},... et s_{k1},s_{k2},... Les trajectoires discrètes figurent en pointillés.

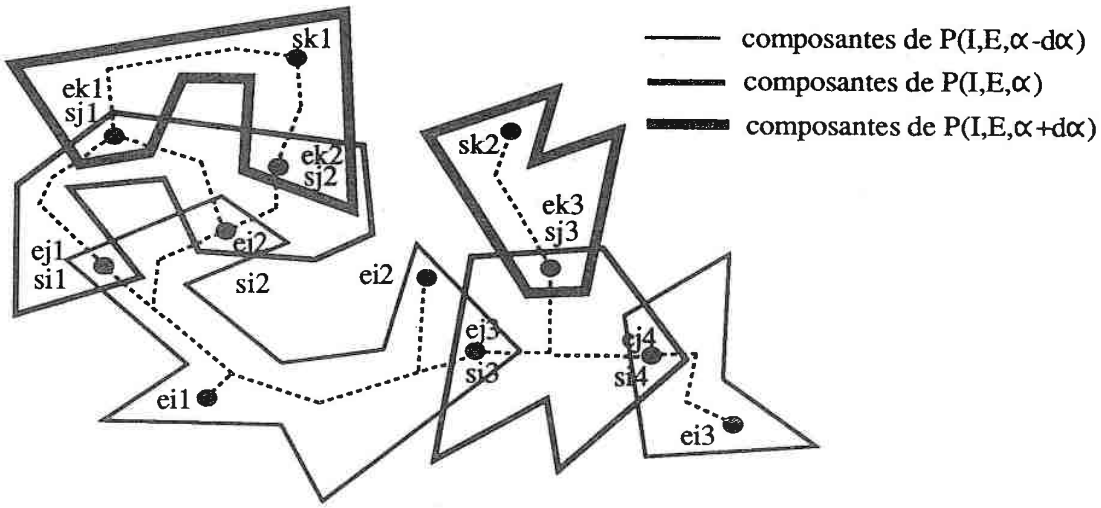


Figure 13 : Points de communication entre les composantes et trajectoires élémentaires entre ces points

Après avoir défini ce que seront les schémas élémentaires de déplacement dans le modèle final, nous pouvons préciser l'algorithme de détermination d'un point de communication au sein d'une certaine intersection.

Détermination d'un point de communication Ce point représente en fait le point crucial de l'application. En effet, touts les déplacements élémentaires se feront via ces points. Si ceux-ci sont mal disposés alors ils constitueront des détours obligés qui donneront des trajectoires imparfaites. Nous ne sommes pas parvenu à une solution totalement satisfaisante. Une qualité évidente des points de communication est qu'ils doivent être éloignés le plus possible des bords des composantes connexes si l'on désire éviter des déplacements en contact avec l'environnement. Nous avons choisi une règle très simple pour la détermination d'un point de communication. Soit R le rectangle englobant la composante. Soit l la droite discrète parallèle à l'axe des j et passant par le centre du rectangle. Nous retiendrons comme point de communication, la cellule située au milieu de la CCL la plus large de la ligne l. Voir figure 14.

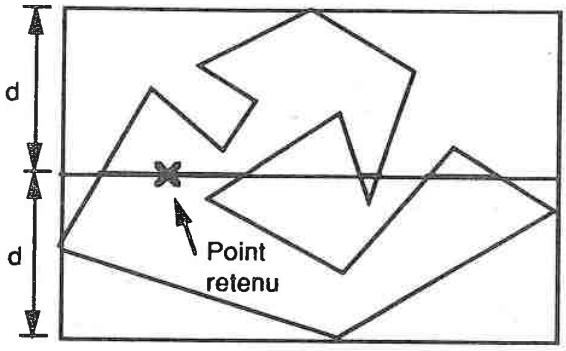


Figure 14 : Détermination d'un point de communication dans une composante

Cette méthode présente 2 inconvénients :

• Elle est peu stable : une petite modification dans la composante peut entraîner un choix très différent. Ce phénomène risque de se produire car les composantes varient peu entre 2 tranches consécutives du modèle. La figure 15 donne un exemple d'un tel phénomène.

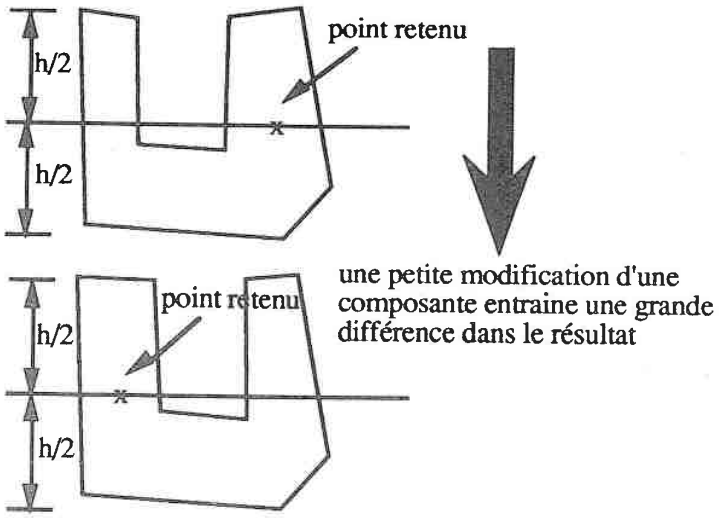


Figure 15 : Instabilité de la méthode de choix d'un point de communication

• Le second inconvénient est du même ordre : Il se peut que le choix d'un certain point de communication entraîne un détour important dans la trajectoire élémentaire qui empruntera ce point (voir la figure 16). Une solution simple peut consister à créer plusieurs points de communication dans une composante.

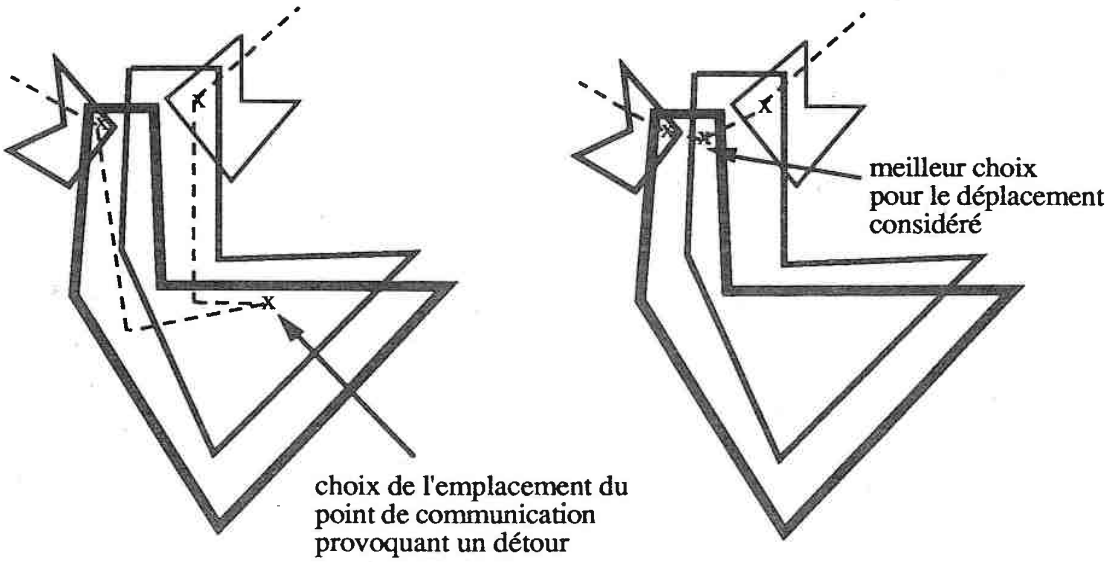


Figure 16 : Trajectoire détournée par le choix d'un point de communication

Algorithme de contruction du modèle de l'espace libre

Le modèle de l'espace libre sera un graphe tridimensionnel dont les noeuds seront des points de communication au niveau desquels vont s'effectuer les rotations et dont les arcs sont les translations élémentaires entre les points de communication. Les arcs se présentent sous la forme de trajectoires discrètes. Les noeuds assurent les communications entre les composantes connexes des P(I,E,α). Chacun d'eux est l'origine d'un certain nombre d'arcs pour circuler dans la tranche de plus grand alpha (on parle des arcs supérieurs) ainsi que d'un certain nombre d'arcs pour circuler dans la tranche de plus petit alpha (on parle des arcs inférieurs).

Cette construction est particuliérement simple. Nous disposons des outils pour la réaliser.

Fonction de service : La fonction Point_Internes(Liste_Cp_Cnxe_Inter) rend pour chacune des composantes connexes de la liste en entrée, un point interne qui sera ensuite un point de communication. Elle rend aussi le nombre de points internes (égal au nombre d'éléments dans la liste en entrée).

L'algorithme prend en entrée :
- la liste des polygones constituant l'environnement.
- Le polygone I à déplacer.
- la valeur de dα (appelée Incr_Alpha)

L'algorithme rend en sortie

• Un modèle de l'espace libre

Début

/ partie initialisation = calcul des points d'entrée dans la couche Incr_Alpha */*

Calcul dans Liste_Cp_Cnxe_Inf de la liste des composantes connexes de P(I,E,0).

Soit Nb_Cp_Cnxe_Inf ce nombre de composantes connexes.

Calcul dans Liste_Cp_Cnxe_Sup de la liste des composantes connexes de P(I,E,Incr_Alpha).

Soit Nb_Cp_Cnxe_Sup ce nombre de composantes connexes.

Pour i de 1 à Nb_Cp_Cnxe_Inf , Pour j de 1 à Nb_Cp_Cnxe_Sup Répéter

Inter = Liste_Cp_Cnxe_Inf[i] ∩ Liste_Cp_Cnxe_Sup[j]

Si Inter ≠ Ø alors

Calcul des composantes connexes de Inter dans Liste_Cp_Cnxe_Inter

Liste_Entrée, Nb_Pt_Interne_E = Point_Internes(Liste_Cp_Cnxe_Inter)

Fin pour j, Fin pour i

/ fin de l'initialisation. Itération sur toutes les couches et calculs de chemins entre des points de */*

/ communication */*

*Alpha = 2*Incr_Alpha*

Tant que Alpha < 2π répeter

Liste_Cp_Cnxe_Inf = Liste_Cp_Cnxe_Sup

Nb_Cp_Cnxe_Inf = Nb_Cp_Cnxe_Sup

Calcul dans Liste_Cp_Cnxe_Sup de la liste des composantes connexes de P(I,E,Alpha).

Soit Nb_Cp_Cnxe_Sup le nombre de composantes connexes.

/ calcul des points de communication des intersections entre les composantes des 2 */*

/ couches successives */*

Liste_Sortie,Nb_Pt_Interne_S = Liste_Vide,0

Pour i de 1 à Nb_Cp_Cnxe_Inf , Pour j de 1 à Nb_Cp_Cnxe_Sup Répéter

Inter = Liste_Cp_Cnxe_Inf[i] ∩ Liste_Cp_Cnxe_Sup[j]

Si Inter ≠ Ø alors

Calcul des composantes connexes de Inter dans Liste_Cp_Cnxe_Inter

Liste_Sortie, Nb_Pt_Interne_S=

ajout(Liste_Sortie,Nb_Pt_Interne_S,Point_Internes(Liste_Cp_Cnxe_Inter))

fsi

Fin pour j, fin pour i

/ calcul des chemins entre les points d'entrée et les points de sortie */*
Pour i de 1 à Nb_Pt_Interne_E , Pour j de 1 à Nb_Pt_Interne_S Répéter
Si Liste_Entrée[i] et Liste_Sortie[j] appartiennent à la même
composante connexe de Liste_Cp_Cnxe_Inf alors
Créer un chemin entre ces 2 points
Ajouter chemin au modèle
Fin pour j, Fin pour i
/ les futurs points d'entrée sont les points de sortie actuels */*
Liste_Entrée,Nb_Pt_Interne_E = Liste_Sortie, Nb_Pt_Interne_S
Fin tant que
fin

Remarque Cet algorithme permet de ne bâtir le modèle que pour un certain intervalle angulaire de $[0,2\pi[$. On perd dans ce cas la notion d'exactitude (à la discrétisation près).

Complexité

On supposera que toutes les opérations booléennes se font entre des matrices comportant P pixels. Le coût d'une opération booléenne (intersection, union, etc...) est donc O(P).
Soit C le nombre de "tranches" du modèle. L'itération principale est exécutée C fois.

Pour chaque itération nous avons :

• **Le calcul de la liste des componsantes connexes d'un certain P(I,E,α)**. Il y a m.n parallélogrammes à retirer à une certaine région. Le coût du retrait d'un parallélogramme est de O(P) donc le calcul de la liste des composantes connexe coûte O(P.m.n) (en négligeant les coûts d'accés dans la table des équivalences).

• **Le calcul des points de communication entre 2 "tranches"**. Il ne peut y avoir plus de composantes que de double-contacts car chaque sommet d'une composante est labellée par un double-contact. Il y a au pire $O(m^2.n^2)$ double-contacts [A & B 88]. Le calcul d'une intersection entre 2 composantes coûte O(P) donc le coût du calcul des points de communication vaut $O(P.m^4.n^4)$.

• **Le calcul des chemins entre les points de communication appartenant à une même composante connexe**. Pour chaque point de communication en entrée, il faut chercher parmi ceux en sortie ceux appartenant à la même composante. Un accés dans la liste des points en sortie coûte O(log(m.n)) vu le tri préalable de cette liste(coût en $O(\log(m.n).m^4.n^4)$)). Nous

faisons la supposition que le nombre de points en sortie appartenant à la même composante qu'un certain point en entrée est constant. Vu les $O(m^4.n^4)$ points possibles en entrée nous avons donc une itération qui coûte $O(\log(m.n).m^4.n^4)$. Le coût de la constitution d'un chemin entre 2 points est proportionnel à la longueur du chemin. Supposons que ce coût est borné par la constante L. Finalement le coût global pour la constitution de l'ensemble des chemins est en $O(L.\log(m.n).m^4.n^4)$.

Conclusion Les complexités exprimées ici sont des complexité dans le pire cas. Elles sont peu significatives lorsque l'on observe la réalité des objets présents. Une étude en moyenne reste à faire... C'est l'objet d'un travail en cours.
Dans l'itération générale, le coût le plus important provient de la constitution des chemins. Nous avons donc une complexité globale en $O(M.L.\log(m.n).m^4.n^4)$.

Il ne reste plus qu'à réaliser effectivement le déplacement demandé. Celui-ci se réduira à la recherche d'un chemin dans le graphe tridimensionnel du modèle. Soient $(X_{dep},Y_{dep},\alpha_{dep})$ et $(X_{fin},Y_{fin},\alpha_{fin})$ les configurations extrêmes du déplacement (adep et afin appartiennent à la suite discrète des α). La première phase de l'algorithme consiste à localiser (X_{dep},Y_{dep}) dans $P(I,E,\alpha_{dep})$. Il suffit de construire $P(I,E,\alpha_{dep})$ et de réaliser une décomposition en composantes connexes pour localiser (X_{dep},Y_{dep}). Cette localisation fournit aussi un point de communication ES_Dep de la composante à laquelle appartient (X_{dep},Y_{dep}). Ce point de communication constituera le point de départ véritable de la recherche d'un plus court chemin dans le modèle. A l'aide de l'algorithme de création d'un chemin dans une composante, I est déplacé entre (X_{dep},Y_{dep}) et ES_Dep. Il suffit de répéter symétriquement ces opérations sur le point (X_{fin},Y_{fin}). Sa localisation fournit un point de communication ES_Fin. Un chemin est recherché entre ES_DEP et ES_Fin. S'il existe, le déplacement est effectué. Il ne reste plus qu'à réaliser le déplacement entre ES_Fin et (X_{fin},Y_{fin}).
L'algorithme s'écrit :

En entrèe :
- Le modèle de l'espace libre
- Les configurations extrêmes $(X_{dep},Y_{dep},\alpha_{dep})$ et $(X_{fin},Y_{fin},\alpha_{fin})$.

En sortie :
- Une suite de déplacements élémentaires aux extrémités desquels s'effectuent des rotations.

Début
Calcul dans Liste_Cp_Cnxe_Inf de la liste des composantes connexes de $P(I,E,\alpha_{dep})$.
Soit Nb_Cp_Cnxe_Inf ce nombre de composantes connexes.

Déterminer à quelle composante connexe appartient (X_{dep},Y_{dep}).
Choisir un point d'entrée ou de sortie dans cette composante : ES_Dep
Créer un chemin entre (X_{dep},Y_{dep}) *et ES_Dep*
Déplacer I suivant ce chemin

Calcul dans Liste_Cp_Cnxe_Sup de la liste des composantes connexes de $P(I,E,\alpha_{fin})$.
Soit Nb_Cp_Cnxe_Sup ce nombre de composantes connexes.
Déterminer à quelle composante connexe appartient (X_{fin},Y_{fin}).
Choisir un point d'entrée ou de sortie dans cette composante : ES_Fin

Déterminer à l'aide d'un algorithme de type A un plus court chemin, dans le modèle, entre ES_Dep et ES_Fin.*
Déplacer I suivant ce chemin

Créer un chemin entre ES_Fin et (X_{fin},Y_{fin})
Déplacer I suivant ce chemin
fin

Remarque La valuation d'un arc du graphe vaut la somme des translations qu'il contient. Il est néanmoins possible d'affecter un coût aux rotations (positif ou négatif) et de favoriser ainsi soit les translations soit les rotations.

Complexité La complexité de l'algorithme de déplacement est celle de l'algorithme A* recherchant une trajectoire dans le graphe c'est-à-dire O(M.log(N)) où M est le nombre d'arcs du modèle et où N est le nombre de sommets. Nous avions établi que le nombre de points de communication étaient en $O(m^4.n^4)$ ainsi que le nombre d'arcs si on suppose qu'un point de communication possède un nombre constant d'arcs. La complexité de l'algorithme de déplacement est donc en $O(m^4.n^4.\log(m.n))$.

Conclusion L'algorithme proposé a été implanté en langage C et donne de bons résultats. Sa sécurité s'est effectivement avérée bonne étant donné la simplicité des opérations de base. Son point crucial reste la détermination des points de communication. Certaines trajectoires sont entachées de détours. Ce point reste à améliorer.
Il existe une possibilité de généralisation. Nous avons introduit au chapitre 4 la définition générale d'une discrétisation. En particulier, les notions de voisinage (8-connexité dans $\mathbb{Z}^2$, 26-connexité dans $\mathbb{Z}^3$) se généralisent. Une décomposition en composantes connexes est possible. La contre-indication vient bien sûr de la nature exponentielle des algorithmes...

Nous proposons en §5.5.3 un algorithme général (mais coûteux) permettant de calculer P(I,E,α) pour des régions de formes quelconques.
Les coûts mémoire se sont révélés maîtrisables. Les figures 17 et 18 donnent 2 copies d'écran.

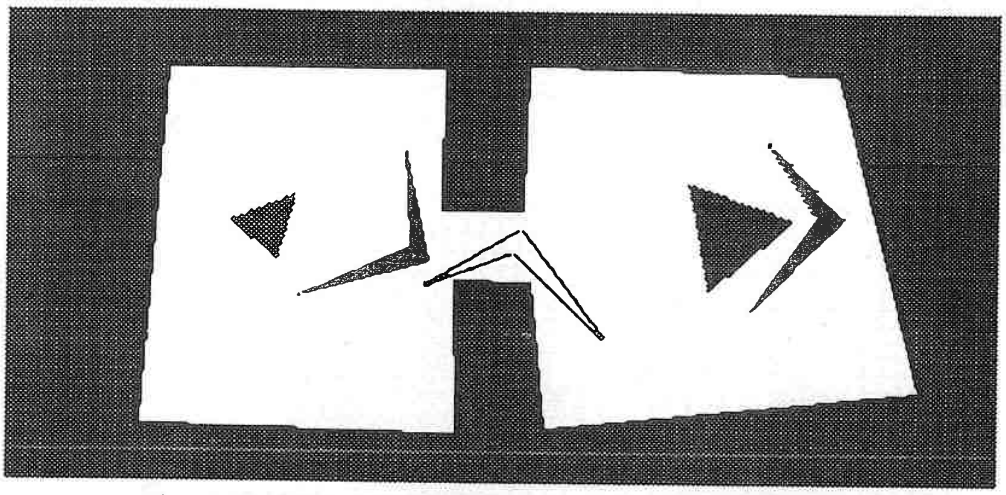


Figure 17 : Déplacement d'un polygone (traits simples) entre
les configurations extrêmes marquées en noir (déplacement de la droite vers la gauche)

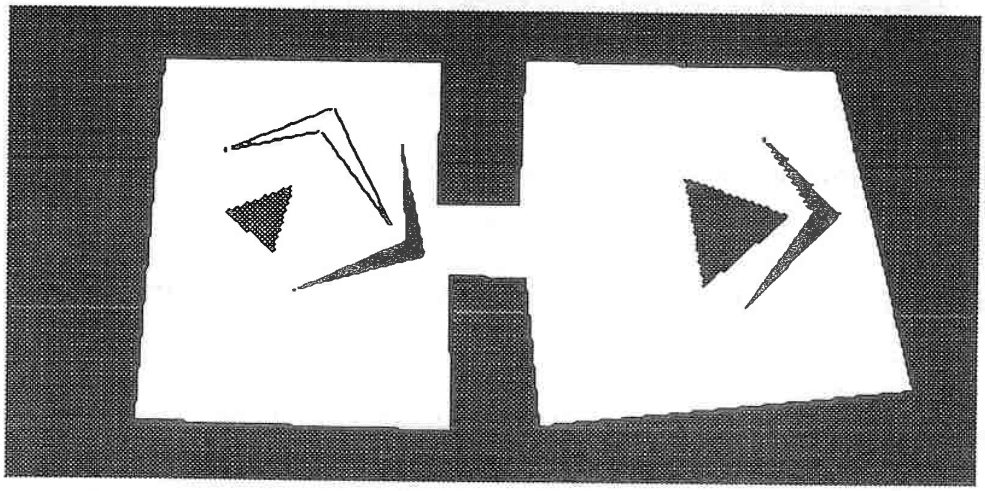


Figure 18 : autre situation. Le polygone est proche de la configuration
finale. La rotation finale s'effectue loin des obstacles

5.5.3 Calcul de P(I,E,α) pour un environnement et un système robotique de formes quelconques

Une région modélise un ensemble quelconque de cellules. Il est donc possible de représenter les obstacles par une suite de régions quelconques. Soit I le système robotique représenté par une

région. Soit C_0 une cellule de référence de I. L'algorithme que nous proposons ici vient de la définition même de $P(I,E,\alpha)$. Soit R un rectangle englobant la scène.

Nous avions : $P(I,E,\alpha) = \mathbf{U}\ (i,j) \in R$ tels que $P_{(i,j)}(I) \cap E = \varnothing$.

Il suffit donc de réaliser tous les placements possibles de I suivant les cellules de R et de retenir ceux donnant une intersection vide (voir figure 19).

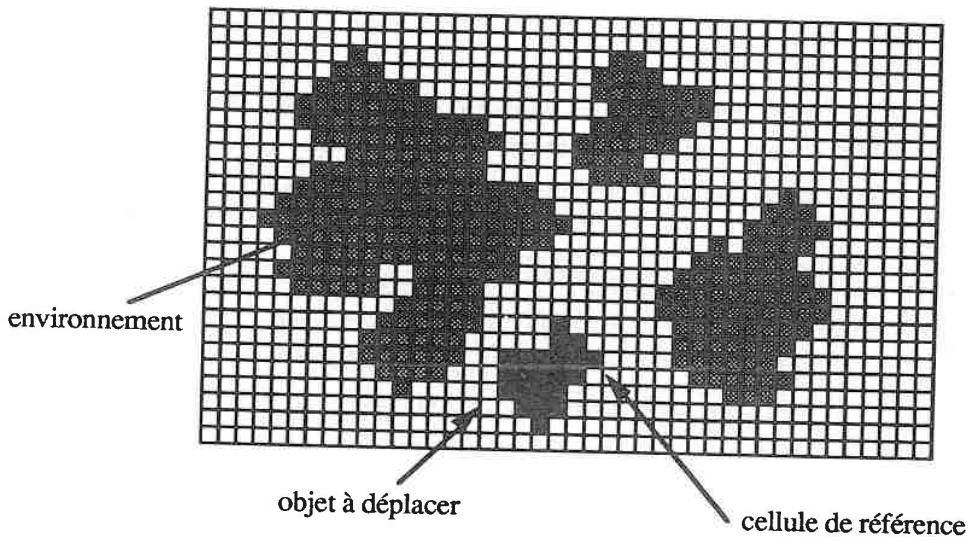


Figure 19 : Calcul de l'espace libre dans le cas de formes quelconques

5.6 Application au déplacement sans collision dans un univers inconnu

Nous proposons ici un algorithme général de recherche de trajectoires dans un univers inconnu. Ce travail n'a pu être validé par une implémentation. Il existe peu de recherches concernant la prise en compte d'un environnement inconnu mais il apparait évident que cette lacune se comblera rapidement dans les années à venir. Voir néanmoins [RIO 88], [L & S 86], [O & I 87].
On peut distinguer 2 axes de recherche : Le premier consiste à acquérir le maximum d'information concernant l'environnement puis à rechercher une trajectoire dans un environnement connu ou alors de réaliser la prise en compte de l'environnement en même temps que l'on déplace le robot. Les outils de la géométrie discrète vont s'avérer ici très puissants. En particulier, les mises à jour dynamiques de l'environnement sont aisées puisqu'il s'agira de réaliser des unions successives.
Pour notre part, la recherche d'une trajectoire sera conjointe à la prise en compte de l'environnement.

La définition générale du problème est identique à celle donnée en tête de ce chapitre.
L'idée générale est la suivante : le système robotique I suivra le diagramme de Voronoï de la portion visible de l'environnement. A chaque sommet de Voronoï découvert, il explorera l'arc se rapprochant le plus du but. Les mises à jour de l'environnement connu se feront pour chaque position discrète atteinte. La difficulté principale de cette étude est donc la création du diagramme de Voronoï. La définition sous-jacente, utilisée ici, du diagramme de Voronoï est celle qui dit qu'il est constitué par l'ensemble des points à égale distance d'au moins 2 objets et que ces 2 objets sont 2 plus proches objets.
Donnons tout d'abord quelques définitions générales :

Définition L'algorithme bâtit progressivement un modèle discret de l'environnement. Nous supposerons qu'il dispose d'une mémoire discrète suffisament grande pour contenir une représentation de l'environnement. Nous appellerons R cette mémoire.

Définition Nous supposerons que le système robotique I est doté d'un organne percepteur, équivalent à un point, capable de rendre à tout moment la région discrète équivalent à sa visibilité dans l'environnement. Nous noterons $\Omega_{i,j}$ cette région visible depuis le point (i,j). Nous supposerons qu'à tout moment, le robot connaît sa position dans un repère fixe (i,O,j) associé à l'environnement (donc aussi associé à la mémoire de travail R).

Ce système peut être de type radar ou à balayage à l'aide d'un faisceau laser. Voir la figure 20.

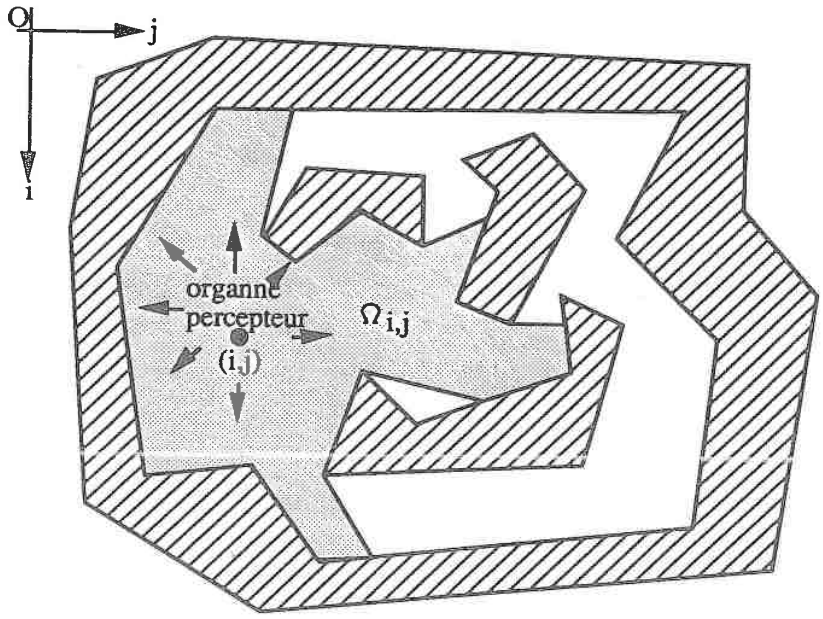


Figure 20 : Définition de Ω_{ij}

Définition Nous introduisons la notion de temps. Notons T_t l'ensemble des cellules discrètes atteintes au temps t par l'organe percepteur du robot. L'environnement connu à cet instant sera noté Ω_t et vaudra : $\Omega_t = \bigcup_{(i,j) \in Tt} \Omega i,j$.

Cette union sera réalisée par une suite d'unions de régions binaires à l'aide de l'algorithme définit précédemment.

Définition L'espace inconnu au temps t est constitué par :
- Les obstacles (par définition)
- L'espace non visualisé par l'organe percepteur du robot au temps t.

Nous le noterons EI_t et il vaudra : $EI_t = R - \Omega_t$ où - signifie différence ensembliste.

Remarque Il y a au moins autant d'obstacles que de composantes connexes dans EI_t.

Définition Soit (i,j) une cellule de Ω_t. Soit $D_r(i,j)$ le disque discret de centre (i,j) et de rayon r. On appellera $D_r(EI_t,(i,j))$ l'intersection entre le disque discret de centre (i,j) et de rayon r avec EI_t. Soient Cp1,Cp2,...,Cps les s composantes connexes de $D_r(EI_t,(i,j))$. On introduit une numérotation des composantes (issue de l'algorithme de décomposition en composantes connexes). Les cellules d'une même composante seront affectées d'un même numéro (voir figure 21).

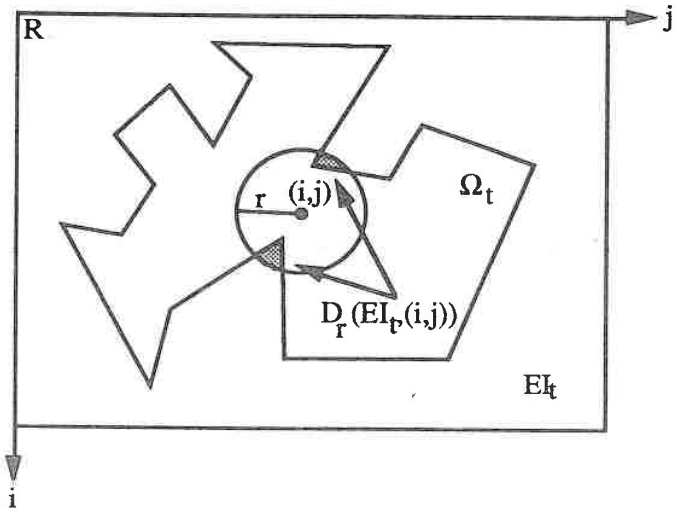


Figure 21 : Définition de $D_r(EI_t,(i,j))$

Définition Soit (i,j) une cellule de Ω_t. On appellera $S(EI_t,(i,j))$ le plus petit disque discret centré en (i,j) tel que $S(EI_t,(i,j)) \cap EI_t \neq \varnothing$. On appellera Cmin(i,j) la première cellule

apparaissant sur le bord de $S(EI_t,(i,j)) \cap EI_t$ si on considére un sens CW et une origine à midi sur le cercle (voir figure 22).

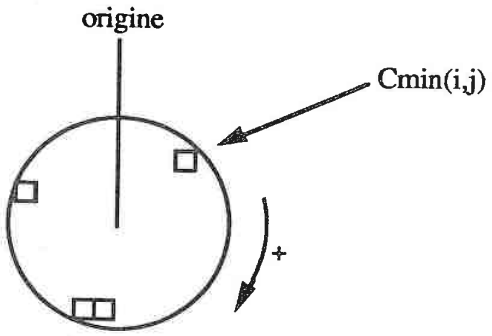


Figure 22 : Définition de Cmin(i,j)

En fait, nous faisons tout pour nous rapporter aux idées définies au chapitre 3 pour la création d'un diagramme de Voronoï discret. Nous disposons désormais des outils nous permettant d'utiliser la notion d'influence d'une cellule. Rappelons que lorsque l'environnement était connu, nous appelions influence d'une cellule, le plus proche objet de cette cellule. Dans ce contexte, nous aurons :

Définition Soit (i,j) la dernière cellule trouvée sur le diagramme de Voronoï. (i,j) possède 8 voisins à examiner. Soit $D_r(i,j)$ le disque centré en (i,j) et tel que $D_r(EIt,(i,j))$ contienne tous les $S(EI_t,(i1,j1)) \cap EI_t$ des voisins (i1,j1) de (i,j) ainsi que des cellules gauches et hautes des voisins (intuitivement, r est suffisamment grand pour que $D_r(EIt,(i,j))$ contienne toutes les cellules considérées comme plus proches des voisins de (i,j) et comme plus proches des cellules gauches et hautes des voisins). Nous appellerons influence de (i1,j1), (i1,i1) voisin de (i,j), le numéro de la cellule Cmin(i1,j1) (voir figure 23).

Identiquement à la définition discrète que nous avons déja donnée, le voisin (i1,j1) appartiendra au diagramme de Voronoï si Influence(i1-1,j1) ≠ Influence(i1,j1) ou si Influence(i1,j1-1) ≠ Influence(i1,j1).

La difficulté réside ici dans la détermination de r. Remarquons que d'une position du robot à l'autre, r varie peu. Il augmente lorsque la distance aux obstacles augmente et vice versa. Nous pouvons désormais écrire l'algorithme. Il est constitué par le mariage de 2 algorithmes que nous avons déja écrits : l'algorithme de constitution discrète du diagramme de Voronoï et celui de cheminement le long d'une trajectoire discrète. L'environnement vaut ici, au temps t, EI_t.

Algorithme

En entrée :

• Le système robotique I doté d'un organe percepteur. Cet organe occupera la position du point T dans le repère lié au robot.

• Un environnement inconnu (!)

• Les 2 configurations extrêmes de la trajectoire recherchée.

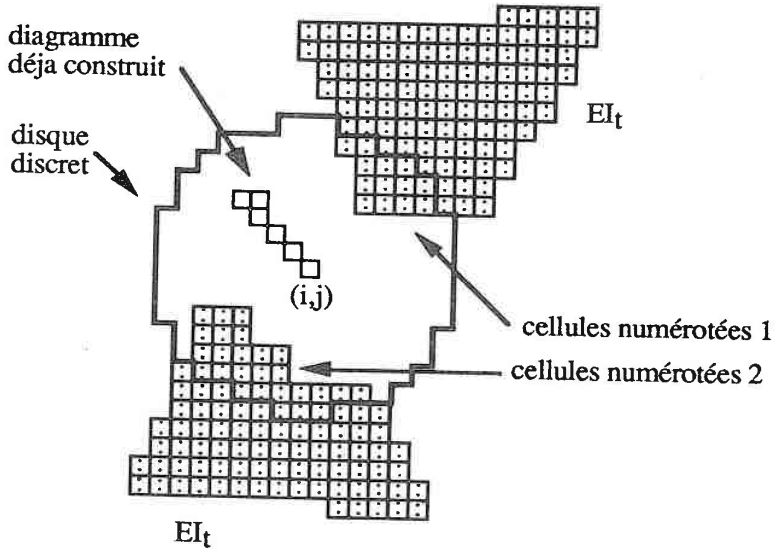


Figure 23 : Influence d'une cellule dans un univers inconnu

En sortie :

• Une trajectoire discrète

Début

Initialisations

Tant que Non_Vide(Pile) et non(But_Atteint) Répéter

- */* soit (nx,ny,nα) la configuration actuelle du robot */*
- */* soit (i,j) la position actuelle de l'organe percepteur du robot */*
- *(Px,Py) ← Dépiler(pile)*
- *si (Px,Py) non voisin de (i,j)*

  alors Reculer le robot sur le diagramme jusqu'à la cellule (i,j) à partir de laquelle (Px,Py) avait été déterminé.

/ ceci équivaut à reculer jusqu'au dernier sommet de Voronoï trouvé */*

• Calculer le point Q situé à la distance F de (Px,Py)

• Soit (mx,my,mα) la configuration du robot suivant les paramètres T (positionné en (px,py)) et Q.

• Si P(mx,my,ma)(I) ∩ EI_t ≠ ∅

alors • si collision de type bas alors .... idem algo chapitre 4

sinon si collision de type queue

alors ... idem algo chapitre 4

• Si non échec de la résolution des collisions

alors Empiler(pile,(px,py))

sinon • Créer le segment discret dans EC entre les configurations (nx,ny,nα) et (mx,my,mα) et effectuer le déplacement correspondant

• Mettre à jour le graphe de Voronoï associé

• Mettre à jour Ω_t et EI_t

• Calculer Dr(EI_t,(Px,Py)) et ses composantes connexes numérotées

• Liste ← Liste_Vide

• Pour (r,s) dans voisin de (Px,Py), n'appartenant pas déja au diagramme et tel que Influence(r-1,s) ≠ Influence(r,s) ou Influence(r,s-1) ≠ Influence(r,s) Répéter ajouter_Liste(Liste,(r,s))

• Trier les voisins dans Liste par ordre croissant de rapprochement du but et les empiler de telle manière que le voisin donnant un arc se rapprochant le plus du but soit en tête de pile.

fsi

fin tant que

Fin

Complexité La complexité est aisée à calculer puisqu'elle est presque identique à celle de l'algorithme de création du diagramme discret. A chaque pas de l'algorithme, il est nécessaire de mettre à jour l'environnement à l'aide d'une union entre Ω_{t-1} et $\Omega(i,j)$. La complexité est donc multipliée par le coût de cette mise à jour. Le coût des détections des intersections peut être considéré comme constant puisqu'à tout moment, une collision est détectée par le calcul de l'intersection entre I et EI_t.

5.7 Conclusion

Les objets discrets et les algorithmes les manipulant se sont avérés simples d'emploi et puissants. Ils nous ont permis de réaliser un algorithme de planification de trajectoire alors que jusqu'ici, ils n'étaient utilisés qu'en traitement d'image. Les opérations dicrètes se sont avérées être

particulièrement puissantes dans les mises à jour dynamiques et dans la traduction de l'idée de localité.

Chapitre 6

Conclusion et perspectives

Nous avons présenté dans cette thèse un certain nombre de méthodes et d'algorithmes pour la planification de trajectoires. Nous avons surtout mis l'accent sur des solutions pratiques mais nous avons aussi proposé un algorithme exact pour le déplacement d'un polygone en translation et rotation. Nous avons pourtant l'intuition que les algorithmes exacts (dans leurs formes actuelles) n'offrent pas une réponse satisfaisante à la planification de trajectoires :

• Ils ne peuvent prendre en compte qu'un nombre réduit de degrés de liberté. La solution générale de Collins ou de Canny est exacte mais sa complexité exponentielle interdit toute implantation pratique. Il est seulement possible de résoudre des cas particuliers (3 degrés de liberté pour le polygone en translation et rotation). A notre connaissance, il n'existe pas d'algorithme exact pour la planification de trajectoires d'un système robotique à 4 degrés de liberté.

• Ils n'offrent pas de qualités dynamiques. Toute modification de l'environnement ou du système robotique est difficile à répercuter sur le modèle sans recalcul complet. Or cet aspect est primordial dans les applications pratiques. Actuellement, des efforts importants sont consentis pour explorer cette voie.

• Ils sont inutilement puissants. Ils offrent un niveau de détail le plus souvent inutilisé. Il serait souhaitable que les algorithmes actuels travaillent à 2 niveaux : à un niveau global général où les détails fins de l'environnement ne sont pas pris en compte et à un niveau local où des procédures fines entreraient en jeu. De plus, une trajectoire à priori réduite (configurations extrêmes proches) entraîne le calcul de tout l'espace libre alors que seule une petite partie est utile

• Ils ont par contre l'avantage de faire prendre clairement conscience des complexités sous-jacentes. Nous savons par exemple, que l'espace libre pour un polygone en translation et rotation est un objet mathématique de complexité $O(m^3.n^3)$ dans le pire cas.

Nous avons donc porté nos efforts sur des algorithmes à base d'heuristiques (cheminement sur une trajectoire discrète - résolution de collisions). Ces algorithmes ont une complexité en moyenne intéressante mais nous ne maîtrisons pas ce qu'ils ne sont pas capables de faire. Une solution

moyenne a été alors proposée : l'introduction de l'idée d'un univers discret. Ces algorithmes tendent vers des algorithmes exacts quand la discrétisation tend vers 0. Il devrait être possible d'introduire une discrétisation variable suivant les situations. Elle pourrait être grossière lorsqu'on sait à priori que les risques de collisions sont faibles (peu d'obstacles ou déplacement dans une zone importante de Ω) et se raffiner dans les configurations étroites de l'environnement. Les algorithmes discrets ont montré des qualités dynamiques (mises à jour aisées dans le cas du déplacement dans un univers inconnu) et des qualités de localité (collisions détectées dans l'environnement proche du système robotique). Il se sont aussi avérés robustes.

Les perspectives ne manquent pas dans ce domaine :

• La prise en compte d'un environnement inconnu a été très peu étudiée. Tous les résultats actuels ne valent que dans le cas d'un environnement connu. Que devient la complexité générale de la planification de trajectoires dans ce cas ? Nous frémissons d'inquétude avant l'annonce d'un résultat car elle sera certainenement moins bonne que celle déja désastreuse dans le cas d'un environnement connu. Les outils de la géométrie discrète et dynamique se sont révélés puissants et bien adaptés à ce type de situations.

• Les liens avec les systèmes de reconnaissance des formes n'existent pas. Il serait intéressant de coupler un système de compréhension de l'environnement et un système de planification de trajectoire. Nous retrouvons ici un intérêt des données discrètes puisque celles manipulées par les systèmes de reconnaissance de formes le sont. Il y aurait au moins cohérence des données.

• Nous manquons d'outils généraux et automatiques pour l'approche des problèmes. Le projet ELIOS tente cette approche. Il permet un traitement simple des contraintes géométriques d'un certain problème. L'utilisateur fournit en entrée un système d'équations ou d'inéquations caractérisant la demande (une certaine trajectoire recherchée entre 2 configurations extrêmes) et les non collisions. Le système résoud alors les équations/inéquations et répond immédiatement sur la faisabilité du problème. Il propose aussi, s'il a répondu par l'affirmative, une solution. Le sytème peut avoir par exemple la forme suivante :

$$x^3 - y^2 < 3$$
$$x.y > 0$$
$$x^2 + y^2 \geq 1$$
$$x + 3 \neq 0$$

Voir [MRS 90] pour plus de détails.

Ericson et Yap (voir [E & Y 88]) proposent une approche très intéressante avec leur "éditeur géométrique". Il s'agit d'un outil qui permet de composer des structures géométriques complexes où interviennent des variables et des relations entre sous-ensembles et d'en déduire automatiquement des théorèmes ou de nier ou affirmer des conjectures proposées par l'utilisateur. Les idées générales sont assez semblables à celles développées dans ELIOS.

• Etudier plus précisement les conséquences de l'utilisation des nombres réels dans les algorithmes. Ce problème concerne d'autres domaines que celui de l'algorithmique géométrique. Nous avons rencontré de nombreux problèmes liés à l'utilisation des nombres réels au cours du développement des logiciels.

• Etudier les complexités en moyenne des algorithmes mis en jeu. C'est un travail en cours de réalisation qui a fait apparaître des difficultés d'ordre purement mathématique.

• Développer des algorithmes parallèles. L'implantation du calcul du diagramme de Voronoï discret a été une réussite. La configuration du système à base de transputers reste entièrement à la charge de l'utilisateur (voir un exemple au chapitre 3). Il serait intéressant de rendre cette étape plus ou moins automatique.
La conception d'architectures spécialisées pour la planification de trajectoires est une voie prometteuse. Des données de type binaire devraient faciliter de telles conceptions. Il existe déjà de telles architectures en traitement d'images.

• Les aspects dynamiques sont primordiaux. Il est indispensable actuellement de disposer d'algorithmes ayant des qualités dynamiques. Ils ne seraient, dans le cas contraire, d'aucune utilité dans une quelconque implantation à caractère pratique (voir [BDT 90], [BDSTY 90], [GKS 89]).

• Le diagramme de Voronoï nous est apparu comme un outil remarquable. De nombreuses voies le concernant restent à explorer. Il serait très intéressant de calculer le diagramme de Voronoï d'un ensemble de polygones, pour un certain polygone I, et enfin pour les 3 degrès de liberté de I. Il s'agit d'un objet mathématique complexe qui est l'union en α des diagrammes de Voronoï pour des orientations fixées de I. Les idées sont ici celles de Canny quand il parle de "road map" (voir le chapitre 1 et §3.4). Il serait un outil remarquable en planification de trajectoires puisque d'une

part nous aurions un algorithme exact et que d'autre part les déplacements engendrés seraient idéalement éloignés des obstacles.

Nous voudrions ici faire une remarque d'ordre général sur l'activité scientifique en algorithmique géométrique. Le but poursuivi par de nombreux chercheurs est de chercher par tous les moyens à réduire la complexité de la résolution des problèmes. Il apparaît ainsi des algorithmes que l'on peut baptiser (expression de F. Preparata) de baroques. Ils sont compliqués et restent souvent non implantés. L'intérêt théorique peut à la rigueur se comprendre lorsque d'autres complexités s'y rapportent Par exemple, de nombreux algorithmes ont une complexité fonction de celle du meilleur algorithme de triangulation d'une scène polygonale. Mais lorsque tel n'est pas le cas ... L'implantation est souvent une épreuve de vérité et elle élimine les monstres mal formés. Nous pensons qu'il est nécessaire d'introduire d'autres métriques de qualité que la simple expression de la complexité (surtout quand celle-ci est une complexité dans le pire cas).

BIBLIOGRAPHIE

[A 90] Amet H.
A motion planning software based on bitmap algorithms (preprint).
(préprint)

[ABS 88] Amet H., Boissonnat J.D., Schott R.
Calcul dynamique du diagramme de Voronoï d'un ensemble de segments
Actes des journées de Géométrie Algorithmique, INRIA Sophia-Antipolis, 18-20 juin 1990, 143-153

[A & F 90] Amet H., Filbois A.
Algorithme parallèle du diagramme de Voronoï d'un ensemble de segments
Actes des journées de Géométrie Algorithmique, INRIA Sophia-Antipolis, 18-20 juin 1990 134-142

[A & S 89] Amet H., Schott R.
Helpful Heuristics for Motion Planning
Proceedings of the fourth International Symposium on Computer and Information Sciences, Vol. 2, 1049-1058

[A & S 89] Amet H., Schott R.
Déplacement exact d'un polygone au sein d'un ensemble de polygones
AFCET, Actes du 7ième congrès RFIA, Paris, 29/11/89 au 1/12/89, 749-763

[A & B 88] Avnaim F., Boissonnat J.D.
Polygon placement under translation and rotation.
RAIRO vol 23-1, 1989, 5-28

[A & B 88] Avnaim F., Boissonnat J.D., Faverjon B.
A practical exact motion-planning algorithm for polygonal objects amidst polygonal obstacles.
Proceedings 1988 International Conference on Robotics and Automation 1656,1661

[A & Y 89] Alt H., Yap C.K.
Algorithmic Aspects of Motion Planning : A Tutorial Part 2
Algorithms Review, vol 1, no 2, may 1990, 61-77

[AFKMNSU 90]
Alt H., Fleischer R., Kaufmann M., Mehlhorn K., Näher S., Schirra S., Uhrig C.
Approximate Motion Planning and the Complexity of the Boundary of the Union of Simple Geometric Figures (preprint)

[BDSTY 90] Boissonnat J.D., Devillers O., Schott R., Teillaud M., Yvinec M.
Applications of Random Sampling to On-line Algorithms in Computational Geometry
Rapport de recherche INRIA no 1285, Août 1990

[BDT 90] Boissonnat J.D., Devillers O., Teillaud M.
An On-line Construction of Higher Order Voronoï Diagrams and its Randomized Analysis
Proceedings of the second CCCG, Ottawa 1990, 278-281

[B & LP 83] Brooks R.A.and Lozano-Perez T.
A subdivision algorithm in configuration space for Findpath with rotation.
M.I.T. Artificial Intell. Lab., Rep. AIM-684, Dec. 82.

[CAN 85] Canny J.
A Voronoi Method for the Piano-Movers Problem.
Proc. FOCS-85, IEEE, 530-535, 1985.

[CAN 87] Canny J.F.
A New Algebraic Method for Robot Motion Planning and Real Geometry.
Proc.*F.O.C.S. - 87,I.EEE, 39-48, 1987.*

[CAN 88] Canny J.
Some Algebraic and Geometric Computations in P-Space
Proceedings of the 20th STOCS, 460-467 (1988)

[C & D 87] Chazelle B., Dobkin D.P.
Intersection of Convex Objects in 2 and 3 Dimensions
Journal of the Association for Computing Machinery, Vol. 34, No 1, January 87, 1-27

[C & S 89] Clarkson K.L.,Shor P.W.
Applications of Random Sampling in Computational Geometry
Discrete Computational Geometry 4: 387-421 (1989)

[CKV 87] Clarkson K.L., Kapoor S., Vaidya P.M.
Rectilinear shortest path through polygonal obstacles
ACM 0-89791-231-4/87/0006/0251

[C & R 87] Canny J.F. and Reif J.
New Lower Bound Techniques for Robot Motion Planning Problems.
Proc.*F.O.C.S. - 87,I.EEE, 49-60, 1987.*

[DEH 89] Dehne F.
Computing digitized Voronoï diagrams on a systolic screen and applications to clustering.
Lecture Notes in Computer Science no 401 pp14 to 24
Proceedings of the International Symposium Varna, Bulgaria, May/June 1989, 14-24

[DON 84] Donald B.
Motion Planning with 6 Degree of Freedom.
MIT AI lab TR-791, 1984.

[EOW 84] Edelsbrunner H., O'Rourke J., Welzl E.
Stationing Guards in Rectilinear Art Galeries
Computer Vision, Graphics and Image Processing 27, 167-176, 1984

[ERD 84] Erdmann M.
On Motion Planning with Uncertainty.
MIT AI Lab. TR-810, 1984.

[E & S 89] Evans D.J., Stojmenovic I.
On Parallel Computation of Voronoï Diagrams
Parallel Computing 12 (1989) pp 121-125

[E & Y 88] Ericson L.W., Yap C.K.
The design for LineTool, a geometric editor (preprint)

[F & T 87] Faverjon B., Tournassoud P.
The Mixed Approach for Motion Planning : Learning Global Strategies from a Local Planner
IJCAI 87, Milan, august 1987

[FMRWY 89] Fleischer F., Mehlhorn K., Rote G., Welzl E., Yap C.,
On simultaneous Inner and Outer Approximation of Shapes (preprint)

[FOR 87] Fortune S.
A sweepline algorithm for Voronoï diagram
Algorithmica Vol. 2, no. 2, 1987

[GKS 89] Guibas L.J., Knuth D.E., Sharir M.
Randomized Incremental Construction of Delaunay and Voronoï Diagrams
Proceedings ICALP'90 LNCS

[GOY 90] Goodrich M.T., O'Dunlaing C., Yap C.K.
Constructing the Voronoï Diagram of a Set of Line Segments in Parallel

[HOF 89] Hoffmann C.F.
The Problems of Accuracy and Robustness in Geometric Computation
IEEE, COMPUTER, 31-41, March 1989

[JOE 87] Joe B.
Discrete Beta Spline
Computer Graphics, Vol. 21, Number 4, July 1987

[KHA 86] Khatib O.
Real time obstacle avoidance for manipulators and mobile robots.
Int. Journal of robotics Research, 90-99, 1986

[K & M 78] Khatib O., Le Maitre J.F.
Dynamic Control of Manipulators operating in a complex environment.
Proceedings of the 3rd CISM-IFTOMM, Udine, Italie, 1978

[KRU 56] Kruskal J.B.
On the shortest planning subtree of a graph and the traveling salesman problem.
Proc. ACM 7, 48-50 1956

[KUN 87] Kundu S.
A New O(n.log(n)) Algorithm for Computing the Intersection of Convex Polygons
Pattern Recognition, Vol. 20, No 4, 419-424, 1987

[LAU 87] J.P. Laumond
Obstacle Growing in a non polygonal World.
Information Processing Letters 25 ,41 -50, 1987.

[L & D 81] D. T. Lee, R.L. Drysdale
Generalization of Voronoi Diagrams in the plane.
SIAM J. Computer Vol. 10, No 1, February 1981.

[LEE 83] Lee D.T.
Visibility of a simple polygon
Computer vision, graphics, and image processing 22, 207-221 (1983)

[L & R] Liebling T., Röthlisberger H.
Infographie et applications - MASSON (1988)

[L &S 86] Lumelsky V.J., Stepanov A.A.
Dynamic path planning for mobile automaton with limited information on the environnment
IEEE, Trans. Automat. Contr., Vol. AC31, no 11, 1058-1063, 1986

[L & S 85] Leven D., Sharir M.
An efficient and Simple Motion Planning Algorithm for a ladder Moving in 2-dimensional Space Admist Polygonal Barriers
1985 - ACM 0-89791-163-6/85/006/0221

[LP & W 79] Lozano-Perez T. and Wesley M.
An Algorithm for Planning Collision-Free Paths Among Polyedral Obstacles
Comm. ACM, vol 22, no 10, (oct. 79), 560-570.

[MEH 84] Mehlhorn K.
Data Structures and Algorithms 3 : Multi-dimensional Searching and Computational Geometry
EATCCS - Monographs on Theoritical Computer Science - 1984

[MRS 90] Monfroy E., Rusinowitch M., Schott R.
Spécification Opérationnele de Contraintes Géométriques
Actes des Journées de Géométrie Algorithmique, INRIA Sophia-Antipolis, 18-20 juin 1990 41-50

[M & Y 86] S.R. Maddila and C.K. Yap
Moving a polygon arround a corner in a corridor.
Proceedings of the Second internationnal symposium on computationnal geometry 1986 ACM, 187-192

[MAD 87] S.R. Maddila
Decomposition Algorithm for moving a ladder Among Rectangular Obstacles.
(preprint).

[MMD 89] Mehlhorn K., Meiser S., O'Dunlaing C.
On the Constrruction of Abstract Voronoï Diagrams (preprint)

[MOR 79] Moravec H.P.
Visual Mapping by a Robot Rover.
Proc.6th Int. J. Conf. on Artificial Intelligence, Tokyo, Japan, 1979.

[NAT 86] Natarjan B.K.
On Moving and Orienting Objects.
Cornell University, TR 86-775, (1986).

[N & P 82] Nievergelt J., Preparata F.P.
Plane-Sweep Algorithm for Intersecting Geometric Figures
Communications of the ACM, Oct.1982, Vol. 25, Number 10

[O & I 87] Oomen J.B., Iyengar S.S., Rao N.S.V., Kashyap R.L.
Robot navigation in unknow terrains using learned visibility graphs
IEEE, journal robotics and automat., Vol RA-3, no 6, 672-681, Dec. 1987

[O'DSY 83] O'Dunlaing C., Sharir M, Yap C.K.
Retraction: A new approach to Motion-Planning.
Proc.15th.Annual ACM Symposium on Theory of Computing, Boston, Mass., 1983.

[O'D S Y 84] O'Dunlaing, Sharir M and Yap C.
Generalized Voronoi Diagram for a Ladder: I Topological Considerations.
Tech. Rept. 139, Computer science Dept. Courant institut of Math. Sciences, November 1984, (to appear in Comm. Pure Appl. Math.).

[O'D S Y 87] O'Dunlaing C., Sharir M.,Yap C.
Generalized Voronoi Diagrams for a Ladder: II Efficient construction of the diagram.
Algorithmica 2, 27 à 59, 1987.

[PEP 90] Pépin F.
Thèse d'université intitulée "Squelettisation de régions et mise en correspondance par relaxation : Application à la vision 3D"

[P & S 85] Preparata F.., Shamos M.I.
Computational geometry - An introduction, ADDISON WESLEY (1985)

[P & T 89] Preparata F , Tamassia R.
Fully dynamic point location in a monotone subdivision
SIAM J. Computer Vol. 18, No 4, August 1989, 811-830.

[REI 79] Reif J.
Complexity of the mover's problem.
Proc. 20 th IEEE Symp. FOCS, 421-427, 1979.

[R & S 85] Reif J.and Sharir M.
Motion Planning in the Presence of Moving Obstacles.
Proc. 205th IEEE symp. FOCS, 144 -54, 1985

[REV 90] Reveilles J.P.
Les droites en géométrie discrète.
Actes des Journées de géométrie algorithmique, INRIA Sophia-Antipolis 18-20 juin 90, 87 -99

[RIO 88] Rao N.S.V., Iyengar S.S., Oommen B.J., Kashyap R.L.
On terrain model acquisition by a point robot admmist polyedral obstacles
IEEE journal of robotics and automation, Vol. 4, No 4, august 88

[ROH 88] Rohnert H.
Moving Discs between Polygons
Proc.ICALP-88, LN.C.S 503-515,Springer Verlag 1988.

[SHA 89] Sharir M.
Algorithmic Motion Planning in Robotics
IEEE, COMPUTER, 31-41, March 1989

[S & S 82] Schwartz J. and Sharir M.
On the Piano Movers' Problem, II General techniques for computing Topological Properties of Real Algebric Manifolds.
Computer Science Department, New York University report 41, (1982).

[S & S 82] Schwartz J. and Sharir M.
On the Piano Movers' Problem
I. The case of a two-dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers.
Communication on Pure and Applied Mathematics, Vol XXXVI, 345-398 (1983)

[S & S 88] Schwartz J. and Sharir M.
A survey of Motion Planning and Related Geometric Algorithms
Artificial Intelligence 337, 1988, 157-169

[TON 87] Tombre K.
Thèse INPL de l'université de Nancy 1 - 1987

[TOU 88] Tournassoud P.
Géométrie et intelligence artificielle pour les robots. Ed. Hermès.

[UDU 77] Udupa S.
Collision detection and avoidance in computer controlled manipulators.
Proc. 5 th Int. Joint Conf. artificial Intell.

NOM DE L'ETUDIANT : **AMET Henri**

NATURE DE LA THESE : doctorat de l'Université de Nancy I

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 05 DEC. 1990 n°2583

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT
M. BOULANGE

Résumé

Cette thèse s'intéresse à la recherche d'une trajectoire sans collision pour un système robotique au sein d'un environnement connu à priori ou inconnu. Ce travail s'inscrit dans le cadre général de l'algorithmique géométrique, nouvelle branche de l'informatique qui traite des problèmes de nature géométrique.

Nous présentons au chapitre 1 un rapide aperçu des différents axes de recherche dans le domaine de la planification de trajectoires.

Nous présentons ensuite au chapitre 2 un algorithme exact résolvant le déplacement en translation et rotation d'un polygone quelconque au sein d'un environnement composé de polygones quelconques.

Le chapitre 3 traite d'un outil très utilisé en planification de trajectoires : le diagramme de Voronoï. Nous présentons rapidement les algorithmes existants pour le calcul du diagramme sur des points. Nous nous intéressons ensuite au calcul dynamique du diagramme d'un ensemble de segments. Puis nous présentons des algorithmes concernant le calcul d'un diagramme discret pour différentes configurations : diagramme d'objets quelconques dans le plan et diagramme généralisés au cas d'un polygone au sein d'un ensemble de polygones.

Nous proposons au chapitre 4 une application pratique de l'utilisation des diagrammes de Voronoï : ils fournissent une trajectoire guide pour des déplacements en translation et rotations.

Le chapitre 5 propose une solution algorithmique au même problème que celui traité au chapitre 2 mais dans un univers discret et à l'aide d'opérations spécifiques à cet univers. Nous montrons qu'elles possèdent de nombreux avantages : des qualités dynamiques et locales ainsi qu'une grande sécurité d'emploi.

Les solutions proposées sont originales. L'effort a essentiellement porté sur le developpement de logiciels. Ce travail a fait l'objet de plusieurs publications (voir la bibliographie).

Mots clés :

Géométrie algorithmique, planification de trajectoires, robots mobiles, diagrammes de Voronoï.