Méthodes probabilistes pour la planification réactive de mouvements

Léonard Jaillet

THÈSE

présentée au

Laboratoire d'Analyse et d'Architecture des Systèmes

en vue de l'obtention du

Doctorat de l'*Université Paul Sabatier de Toulouse*
Ecole Doctorale Systèmes

Spécialité : **Systèmes Informatiques / Robotique**

par **Léonard Jaillet**

MÉTHODES PROBABILISTES POUR LA PLANIFICATION RÉACTIVE DE MOUVEMENTS

Soutenue le 19 décembre 2005, devant le jury composé de :

Raja Chatila	*Président*
Oliver Brock	*Rapporteur*
Christian Laugier	*Rapporteur*
Michel Courdesses	*Examinateur*
Steven M. Lavalle	*Examinateur*
Thierry Siméon	*Directeur de thèse*

A la mémoire de mon oncle François

Avant Propos

Je tiens à remercier Malik Ghallab, directeur du LAAS du CNRS, pour m'avoir accueilli dans ce laboratoire. Je remercie également Raja Chatila, directeur du groupe de Robotique et Intelligence Artificielle, de m'avoir permis de travailler dans ce groupe et d'avoir accepté de présider mon jury de thèse.

J'exprime ma sincère reconnaissance à mes deux rapporteurs Oliver Brock de l'université d'Amherst (Massachusetts) et Christian Laugier de l'Inria Rhône-Alpes, pour avoir accepté d'être rapporteurs de mes travaux de thèse ainsi que pour leur lecture pertinente du manuscrit. Merci à Michel Courdesses, pour avoir accepté d'être membre de mon jury de thèse. Un grand merci également à Steve Lavalle, pour avoir accepté de faire partie de ce jury mais aussi pour m'avoir si bien accueilli lors de mon séjour à l'université d'Urbana. Merci à toute son équipe et en particulier à Anna Yershova avec qui les échanges scientifiques furent particulièrement fructueux. Merci également à toute l'équipe de Mark Overmars de l'université d'Utrecht et en particulier à Jur et Dennis, qui ont fait de mon séjour chez eux un grand moment d'échange et de plaisir.

Je voudrais adresser de chaleureux remerciements à mon directeur de thèse Thierry Siméon, qui a su me témoigner toute sa confiance et son soutien au cours de ces 3 années de travail, me guidant à travers les récifs de l'incertitude et autres méandres propres à tout travail de recherche.

Je tiens à exprimer ma sincère reconnaissance à Jean-Paul Laumond, Florent Lamiraux, Juan Cortés et Fabien Gravaux pour leurs inestimables conseils scientifiques et pour leur grande sympathie. Merci à Fred pour sa hotline technique et son amitié sans faille, Nico pour ses olives et son cinq de carreau, merci aussi à tous les autres thésards et nouveaux docteurs qui sont devenus des amis : Manu, Julien, Stéphane, Abed, Aurélie, Claudia, Seb, Max, Alex, Thomas, Olivier, Guillaume et bien d'autres...

Un immense merci à tous les toulousains qui m'ont entouré, soutenu au cours de ces 3 années et qui ont m'ont fait vivre de si bons moments. D'abord la dream team de l'INSERM avec Mélanie, Evelyne, Chloé, Lola et bien sur Gaëlle qui m'a fait entrer dans sa danse et à qui je souhaite le meilleur dans cette nouvelle vie canadienne. Je remercie également Katie, Loïc, pour avoir fait vivre mon appartement la nuit, Hélène, Sophie pour avoir cru à la réussite du cylcotherme et enfin ce cher Robindidon, prince de la nuit à qui je souhaite un grand soleil.

Je voudrais aussi saluer tous mes amis lyonnais, Fanny qui a eu la patience de relire ce manuscrit, Sophie et Valentin, à qui je serre la "pat", Elise et Augustin, pour refaire le monde ou encore le parcourir, Dead, pour une vie de rock'n roll et Marie Poppins cette sorcière au grand coeur. Enfin, toutes mes pensées vont à ma famille, Titou, Marie, Caro, Claude, Lulu, Gaston, Mathéo, Emile, Serge et bien sur mes chers parents.

Table des matières

Introduction

La planification de mouvement en robotique [Lozano-Pérez 83] est un domaine de recherche qui a été largement étudié au cours de ces vingt dernières années [Latombe 91, LaValle 05]. L'objectif de ces travaux est de développer des méthodes algorithmiques à la fois performantes et effectives en pratique qui permettent le calcul automatique de trajectoires pour des systèmes mécaniques, malgré la forte combinatoire de ces problèmes [Schwartz 83b, Canny 88]. Une famille de méthodes très répandue aujourd'hui correspond en particulier aux méthodes probabilistes [Kavraki 96, Svestka 97a] qui, grâce à leur efficacité, ont permis d'étendre le champ d'application de la planification à une grande variété de domaines, allant de la robotique, à la CAO, la bioinformatique ou encore l'animation graphique.

Dans le cadre de cette thèse, on s'intéresse plus particulièrement à ces problèmes dans un contexte réactif c'est à dire dans le cas d'environnements partiellement dynamiques que l'on rencontre dans de nombreuses applications. En effet, très souvent l'environnement peut subir des changements dynamiques : certains objets peuvent être ajoutés ou enlevés de la scène, ou encore changer de position (par exemple une porte qui s'ouvre). Ce type de changement affecte alors les mouvements possibles des entités situées dans l'environnement. Les approches probabilistes sont aujourd'hui les seules méthodes capables de traiter de manière générique des problèmes de planification de mouvement pour des systèmes mécaniques complexes. En l'état, elles sont cependant mal adaptées au cas de scènes partiellement dynamiques. D'un coté les techniques dites "simple requête", qui s'appuient à chaque requête sur une nouvelle structure de données, perdent à chaque fois toute l'information liée aux obstacles statiques. De l'autre, les méthodes à "requêtes multiples", construisent des structures de données statiques qui peuvent être invalidées par les obstacles mobiles. Elles ne peuvent donc pas être utilisées sans l'ajout de mécanismes de mise à jour permettant de tenir compte du contexte courant.

L'objectif de cette thèse est de mettre en place des schémas d'algorithmes probabilistes plus performants pour rendre compte des différents changements susceptibles de se présenter dans

l'environnement. Ces travaux réalisés au sein du groupe de Robotique et Intelligence Artificielle du LAAS-CNRS s'inscrivent dans le cadre du projet Européen MOVIE[1] (Motion Planning in Virtual Environments). Le but de ce projet est de développer de nouvelles techniques permettant de générer automatiquement des mouvements visuellement convainquant, pour de multiples entités autonomes ayant à naviguer dans des environnements virtuels complexes. Dans ce contexte, on s'intéresse donc en particulier aux deux domaines d'application que sont la robotique et la réalité virtuelle.

Les environnements partiellement dynamiques sont très communs en robotique. En milieu d'intérieur, on pourra par exemple posséder le plan d'un bâtiment représentant les éléments fixes (typiquement les murs), auxquels s'ajoutent les éléments dont la position peut davantage varier (tels que le mobilier et les portes). Pourtant la plupart des travaux concernant la planification de mouvement se concentrent soit sur le mouvement de systèmes complexes (e.g. articulés) dans des environnements statiques, soit sur le mouvement de robots mobiles rigides dans des environnements non parfaitement connus, voir dynamiques. Alors que les approches développées pour la navigation réactive de robots nous permettent de prendre en compte une dynamique plus forte que celle considérée dans cette thèse, ces techniques ne sont pas généralisables au cas de systèmes mécaniques plus complexes (bras manipulateurs, mains mobiles). En effet ces systèmes utilisés pour la manipulation d'objets, nécessitent l'exploration d'espaces des configurations plus complexes et pour lesquels les méthodes actuelles ne permettent pas de rendre efficacement compte de changements dynamiques de la scène. Il est donc nécessaire de développer pour la robotique de nouvelles méthodes dédiées au mouvement de systèmes articulés dans des environnements partiellement dynamiques.

Classiquement, dans le domaine de la réalité virtuelle, les méthodes de génération de mouvements combinent des techniques de programmation "en dur" du mouvement qui décrivent de façon précise les divers mouvements possibles des entités virtuelles, avec des techniques issues de l'intelligence artificielle. Ce travail est en général très délicat et requiert des compétences que seul un spécialiste tel qu'un animateur professionnel possède. De plus, comme il est souvent impossible de prévoir l'ensemble des situations que les entités peuvent rencontrer, il arrive que celles-ci adoptent des comportements inattendus et non appropriés par rapport aux situations rencontrées. Par exemple une entité pourra se diriger "obstinément" vers une portion donnée de l'espace, alors que celle-ci n'est pas accessible. Enfin, avec ces méthodes les mouvements générés tendent à être répétitifs et prévisibles, ce que l'on souhaite en général éviter. L'utilisation de méthodes de génération de mouvement plus "évoluées" permettraient aux entités virtuelles de posséder une plus grande autonomie et un comportement global plus réaliste.

Le cœur de notre contribution porte sur la conception d'algorithmes de planification capables de traiter de manière performante ces problèmes de scènes partiellement dynamiques com-

[1] http ://www.give.nl/movie

posées d'une partie statique et d'un ensemble d'obstacles mobiles. Ce planificateur hybride combine deux grandes familles de techniques. D'une part les techniques dites PRM, initialement conçues pour résoudre des problèmes à requêtes multiples et que nous avons étendues à des problèmes de scènes dynamiques. D'autre part, de nouvelles techniques de diffusion, alors que celles-ci sont généralement dédiées aux problèmes à requête simple ne nécessitant aucune opération de prétraitement.

La suite de ce manuscrit s'organise de la façon suivante. Après un bref rappel du formalisme de la planification de mouvement, le chapitre 1 présente les principales méthodes probabilistes de la littérature, et se focalise plus spécifiquement sur les méthodes considérant des environnements dynamiques. Le chapitre 2 expose le principe général d'un planificateur dédié aux environnements à changements dynamiques. La stratégie proposée se base sur la combinaison des méthodes PRM avec des méthodes de diffusion afin d'exploiter leurs avantages respectifs. Ce planificateur repose de plus sur des mécanismes de mise à jour paresseux permettant de rendre l'approche particulièrement efficace. Au cours du chapitre 3, nous présentons une nouvelle méthode de diffusion appelée RRT à Domaine Dynamique. Nous verrons que cette variante de l'algorithme RRT standard s'avère plus efficace dans les cas de problèmes localement très contraints. Dans le cadre des scènes partiellement dynamiques, cette méthode pourra en particulier servir à reconnecter efficacement les portions de structures de données localement invalides. Les chapitres 4 et 5 décrivent deux méthodes de création de réseaux cycliques pouvant être utilisées au cours de la phase d'initialisation du planificateur. Le chapitre 4 présente les réseaux de rétraction qui permettent de capturer à travers une structure de données réduite, les différents types de chemins de l'espace qui sont entre eux "difficilement" déformables. Enfin, le chapitre 5, décrit une méthode permettant d'intégrer la prise en compte des obstacles dynamiques lors de la construction du réseau, afin de garantir un critère de robustesse vis à vis des obstacles mobiles.

Chapitre 1

Formalisme et Approches

La planification de mouvement est un problème classique de robotique qui a suscité de nombreux travaux depuis les travaux pionniers du MIT [Lozano-Pérez 83]. La prise en compte de scènes dynamiques qui fait l'objet de cette thèse correspond à des problèmes plus difficiles que le cas de scènes statiques considéré dans la plupart des travaux. Dans ce chapitre, après une brève introduction à ces problèmes et une synthèse des travaux récents menés à partir des méthodes probabilistes, très populaires aujourd'hui de par leur performance pratique, nous nous focaliserons sur les méthodes développées plus spécifiquement pour la planification en environnement dynamique. Avant de présenter l'état de l'art sur les méthodes rendant compte d'obstacles dynamiques, nous rappelons les éléments de base associés au formalisme d'un problème de planification de mouvement. Pour une formulation plus détaillée du problème et une présentation plus large des extensions et applications, on pourra consulter les ouvrages faisant référence [Canny 88, Latombe 91, Laumond 98a, Gupta 98, LaValle 05].

1 Planification de mouvement

1.1 Données du problème

Un problème de planification de mouvement est défini dans une scène (2D ou 3D) appelée *espace de travail* $\mathcal{W}$ comportant diverses entités géométriques qui peuvent être mobiles (les robots) ou fixes (les obstacles). Une grande variété de représentations peut être utilisée pour décrire la géométrie de ces entités (c.f. [Hoffmann 89, Mortenson 85] pour une description détaillée des méthodes de modélisation géométrique). Chaque entité correspond à un ou plusieurs corps rigides. La localisation spatiale de ces corps (position et orientation) est définie

par une matrice de transformation homogène qui permet de passer d'un repère de référence $R_{\mathcal{W}}$, fixe dans l'espace à un repère $R_{\mathcal{B}}$ associé au corps $\mathcal{B}$. Pour le cas d'un problème 3D, 6 paramètres (3 translations, 3 rotations) définissent cette transformation. La matrice de transformation homogène associée appartient alors au groupe des déplacements $SE(3) = \mathbb{R}^3 \times SO(3)$ où $SO(3)$ désigne le groupe spécial orthogonal des rotations. De nombreux ouvrages de référence en mécanique et en robotique [Paul 81, Craig 89] abordent de façon détaillée le calcul de ces transformations associées aux changements de repères, en fonction de la paramétrisation utilisée pour les rotations (e.g. Euler). Nous présentons maintenant les deux types d'entités d'un problème de planification de mouvement.

Les obstacles

Dans sa formulation classique [Latombe 91] la planification de mouvement considère généralement le cas de scènes composées d'*obstacles statiques* (e.g. les murs d'un bâtiment). Les repères attachés aux corps rigides constituant ces obstacles sont donc fixes. Dans le cadre de cette thèse, nous nous intéressons également aux *obstacles mobiles* dont la position peut varier d'une requête de planification à l'autre. Alors que certains peuvent avoir un nombre discret de placements (typiquement les portes d'un environnement), d'autres peuvent avoir un ensemble continu de deplacements possibles. Certains obstacles peuvent aussi apparaître ou disparaître au cours des requêtes successives.

Les robots

Un robot R représente un système mécanique auquel on associe une représentation cinématique et géométrique. Il est constitué d'un ou de plusieurs solides S_i reliés entre eux par des liaisons cinématiques qui contraignent leur mouvement relatif. L'ensemble de ces liaisons définit la *chaîne cinématique* du robot ainsi que l'ensemble de ses degrés de libertés, c'est à dire, l'ensemble minimal de paramètres nécessaires pour définir parfaitement son état géométrique. Cette chaîne cinématique peut être représentée à l'aide d'un diagramme appelé *diagramme cinématique*. Quand ce diagramme comprend des boucles, on voit l'apparition de nouvelles contraintes qui rendent interdépendantes les différentes liaisons du système. On dit alors que le robot est un mécanisme à *chaîne cinématique fermée*. Notons que la terminologie employée utilise le mot robot, pour désigner une grande variété de systèmes. Un robot pourra tout aussi bien représenter une molécule flexible [Latombe 99], qu'un système articulé comme un acteur virtuel évoluant dans un univers 3D [Pettre 02].

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

La notion d'espace des configurations $\mathcal{C}$ est une notion forte qui permet de formuler de manière générique le problème de la planification de mouvement. Une *configuration* q, définie par un vecteur multidimensionnel, représente l'état géométrique du robot, chaque paramètre du vecteur correspondant à la valeur d'un de ses degrés de liberté. $\mathcal{C}$ représente alors l'ensemble des configurations possibles de q. La force de cette représentation est de pouvoir définir sans

ambiguïté la configuration (position et posture) d'un système articulé quelconque au moyen d'un point unique dans l'espace des configurations. Notons que la dimension de cet espace est égale au nombre de degrés de liberté du système et peut parfois être élevée.

En présence d'obstacles, les configurations qui mènent le robot en collision avec ces obstacles sont interdites. De plus, si le système est articulé, celles qui provoquent une collision avec lui-même, c'est à dire une auto-collision sont également interdites. Une configuration pour laquelle il n'y a ni collision avec les obstacles, ni auto-collision est dite *libre*. L'ensemble des configurations libres est noté $\mathcal{C}_{free}$. Comme les configurations frontières qui correspondent aux positions de contact sont également supposées interdites, $\mathcal{C}_{free}$ est un ouvert. Le complémentaire de $\mathcal{C}_{free}$, qui correspond aux configurations au contact ou à la collision est noté $\mathcal{C}_{obst}$.

Chemins dans $\mathcal{C}$

Comme l'image dans $\mathcal{C}$ du robot est un point, à tout chemin du robot dans $\mathcal{W}$ correspond un chemin représentable par une courbe dans $\mathcal{C}$. Un chemin peut donc être représenté par une fonction continue :

$$\tau : [0,1] \rightarrow \mathcal{C}$$

Un chemin est dit *libre*, si son image par τ est entièrement située dans $\mathcal{C}_{free}$. Un chemin est dit *admissible* s'il respecte les différentes contraintes internes inhérentes à la nature du robot (contraintes de chaîne fermée, contraintes différentielles...). Enfin un chemin est dit *faisable* s'il est *libre* et *admissible*.

1.2 Formulation du problème

Le problème de planification de mouvement "de base" peut être formulé de la façon suivante :
Étant donnés un robot R évoluant dans un espace de travail $\mathcal{W}$ contenant des obstacles, une configuration de départ q_{init} et une configuration d'arrivée q_{goal}, trouver, s'il existe, un chemin faisable reliant q_{init} et q_{goal}. Sinon, signaler qu'il n'existe pas de chemin solution.

Historiquement, ce problème apparaît d'abord sous le nom de problème du *déménageur de piano* [Schwartz 83a]. Du point de vue strictement algorithmique, la décidabilité du problème de la planification de mouvement est résolu [Schwartz 83a, Canny 88]. Cependant les algorithmes proposés ont avant tout un intérêt théorique car ils ne permettent absolument pas de résoudre des problèmes de planification pratiques en des temps raisonnables. En effet, certains résultats relatifs à la complexité sont connus. Ainsi, il a été montré que la planification de mouvement pour un mécanisme articulé composé de polyèdres dans un environnement tridimensionnel est *PSPACE*-difficile [Reif79]. On sait également que l'algorithme exact le plus

efficace est d'une complexité en temps exponentielle selon la dimension de l'espace des configurations [Canny 88]. Ce problème de complexité est un problème majeur inhérent à la planification de mouvement qui se place dans des espaces des configurations qui peuvent être de grande dimension.

Dans les années 80, au lieu d'essayer de caractériser explicitement les frontières de $\mathcal{C}_{obst}$, des méthodes basées sur une discrétisation de $\mathcal{C}$ sont proposées [Faverjon 84, Lozano-Pérez 87]. On dit alors de ces planificateurs qu'ils sont *complets en résolution*, c'est à dire qu'ils garantissent de trouver un chemin solution s'il existe, mais uniquement pour une résolution donnée. Malheureusement, la complexité issue de la dimension de $\mathcal{C}$ demeure. Ainsi, ces planificateurs sont limités à des systèmes comportant peu de degrés de liberté. Autour de la même époque, apparaissent des techniques basées sur des *champs de potentiel* [Khatib 86]. Elles permettent de développer des planificateurs locaux ayant des capacités d'évitement d'obstacles. Ces planificateurs qui s'avèrent efficaces pour certaines classes de problèmes restent malheureusement bloqués dans les situations impliquant des minima locaux. De plus, le choix de la fonction potentielle utilisée s'avère être un point critique souvent difficile à déterminer pour les espaces de haute dimension.

Avec le début des années 90, on voit l'émergence des planificateurs plus généraux, mais satisfaisant une forme de complétude plus faible [Barraquand 91a, Kavraki 96, LaValle 98]. Ces approches utilisent le tirage de configurations spécifiques de l'espace pour casser la complexité propre à la dimension de $\mathcal{C}$. On parle d'*algorithmes probabilistes*. Le terme "probabiliste" caractérise en fait la complétude probabiliste de ces algorithmes qui garantissent de trouver une solution en un temps fini si elle existe. Ces méthodes très efficaces en pratique sont aujourd'hui largement utilisées. La section suivante fait la synthèse de travaux récents sur ces techniques.

2 Algorithmes probabilistes

Les algorithmes probabilistes s'appuient sur l'utilisation de l'aléatoire pour la construction d'un graphe capturant de façon condensée la connexité[1] de l'espace libre $\mathcal{C}_{free}$, permettant ainsi de s'affranchir de toute représentation explicite de régions de $\mathcal{C}$. Généralement, on distingue deux grandes classes de techniques :

– Les méthodes de construction de *réseaux probabilistes* (c.f. figure 1.1 a), appelées aussi méthodes de type PRM (de l'anglais "Probabilistic Roadmap Methods"). Une première phase d'initialisation capture la connexité de l'espace libre. Les requêtes de planification sont ensuite résolues rapidement en se connectant au réseau précalculé. Ce type de méthode est donc particulièrement efficace quand plusieurs requêtes nécessitent d'être résolues pour

[1] On dira que la connexité d'un espace est capturée s'il est possible d'extraire un chemin du graphe reliant toute paire de points appartenant à une même composante connexe.

le même environnement. C'est pourquoi, on dit aussi que ces méthodes sont à *requêtes multiples*.

– Les méthodes *incrémentales* ou de *diffusion* (c.f. figure 1.1 b). Dans ce cas l'exploration se fait en construisant de manière incrémentale des *arbres* de recherche (i.e. des graphes ne contenant aucun cycle) à partir des configurations initiales et but. Ces techniques, contrairement aux précédentes, ne cherchent pas à calculer toute la connexité de $\mathcal{C}_{free}$ mais explorent seulement certaines de ces régions afin de résoudre un problème donné. C'est pourquoi elles sont aussi appelées méthodes à *requête simple*. Notons que l'exploration est implicitement guidée par les nœuds initiaux à partir desquels se développent les arbres. Ainsi, ces méthodes sont généralement plus performantes que celles basées sur les réseaux probabilistes quand les configurations de départ sont situées dans des zones très contraintes.

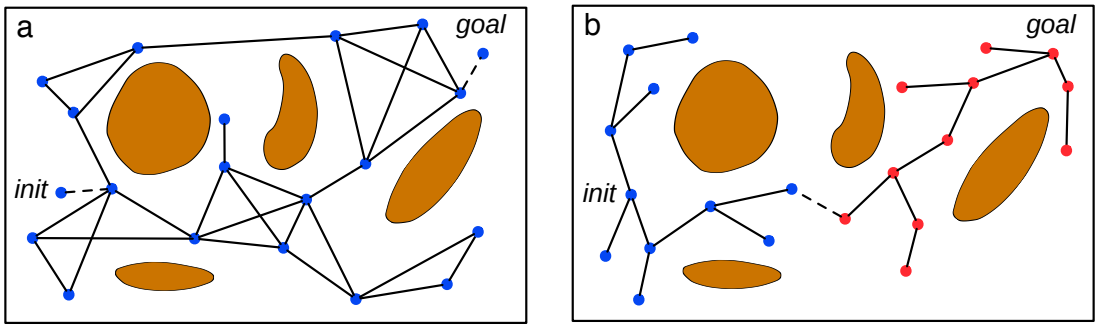


FIG. 1.1 – *On distingue deux grandes familles de méthodes probabilistes : les méthodes de construction de réseaux probabilistes (a) et les méthodes de diffusion (b).*

Nous verrons dans le chapitre 2 comment nous proposons de tirer profit des avantages de ces deux classes de méthodes pour combiner celles-ci au sein d'un planificateur dédié aux environnements changeants.

2.1 Les réseaux probabilistes

Le principe général des méthodes PRM, proposées simultanément à Stanford et à Utrecht [Overmars 95, Kavraki 95, Kavraki 96, Svestka 97a], est le suivant. Les configurations de l'espace sont échantillonnées et celles qui appartiennent à $\mathcal{C}_{free}$ sont conservées comme nœuds du réseau. Ces nœuds sont reliés entre eux au moyen d'arêtes libres qui correspondent à des chemins locaux faisables pour le robot. Une fois ce réseau construit, une requête de planification peut être résolue en connectant les configurations initiales et but au réseau et en recherchant un chemin à travers celui-ci. La recherche peut s'effectuer à l'aide de méthodes classiques d'exploration de graphes tels que l'algorithme de Dijkstra ou encore A^*. Des méthodes qui

permettent d'optimiser et de lisser le chemin trouvé peuvent finalement être appliquées. Nous présentons maintenant les principaux travaux d'extension proposés pour améliorer la performance générale de ces méthodes.

"Passages étroits"

Un des problèmes bien connu, lorsque l'on s'appuie sur un échantillonnage uniforme des configurations, est ce que l'on appelle le *problème du passage étroit*. En effet, il se peut que le réseau calculé ne rende pas compte de la connexité réelle de $\mathcal{C}_{free}$ lorsque certains passages contraints n'ont pas été détectés. Pour pallier cette difficulté, plusieurs travaux ont proposé des variantes afin d'augmenter la densité d'échantillonnage dans des régions critiques, permettant ainsi de mieux détecter ces passages (c.f. figure 1.2).

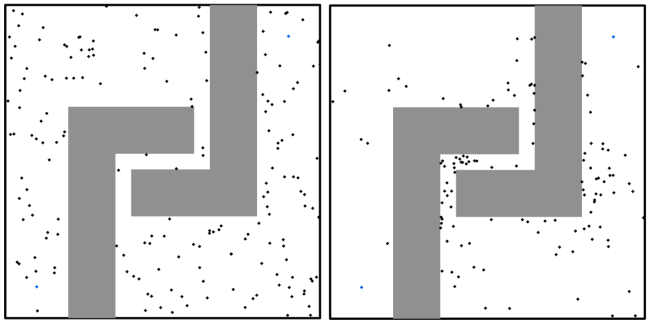


FIG. 1.2 – *Répartition des nœuds dans l'espace pour une méthode de type PRM et une méthode utilisant la notion d'espace dilaté (tiré de [Hsu 98]).*

Une solution proposée [Hsu 98], est de construire initialement un réseau pour un espace libre "dilaté", autorisant une certaine distance de pénétration du robot à l'intérieur des obstacles. Ensuite, ce réseau est modifié en tirant de nouvelles configurations autour des nœuds et des arêtes qui deviennent invalides dans l'espace libre non dilaté. Cette méthode permet d'augmenter les chances de tirer des configurations dans les zones contraintes de $\mathcal{C}_{free}$. L'algorithme OBPRM (pour "Obstacle Based PRM") présenté dans [Amato 98] vise à concentrer des nœuds à la surface des obstacles. Chacune des configurations tirées se trouvant dans $\mathcal{C}_{obst}$ est projetée à l'extérieur des obstacles, en suivant plusieurs directions aléatoirement déterminées. La méthode de projection utilise une technique proche du lancé de rayon appliquée dans le domaine de l'imagerie. Une autre approche basée sur un *tirage gaussien* de configurations [Boor 99] génère directement des points proches de la surface de $\mathcal{C}_{obst}$ (c.f. figure 1.3). Pour cela, des paires de points proches les uns des autres sont d'abord échantillonnées (via une distribution gaussienne). Si l'un des deux points se situe dans $\mathcal{C}_{obst}$ et l'autre dans $\mathcal{C}_{free}$, la configuration libre est conservée. Dans tous les autres cas on rejette les configurations tirées. Une méthode proche de la précédente [Hsu 03] propose un test encore plus sélectif (appelé "bridge test"), qui privilégie les configurations libres entourées par deux configurations en col-

lision.

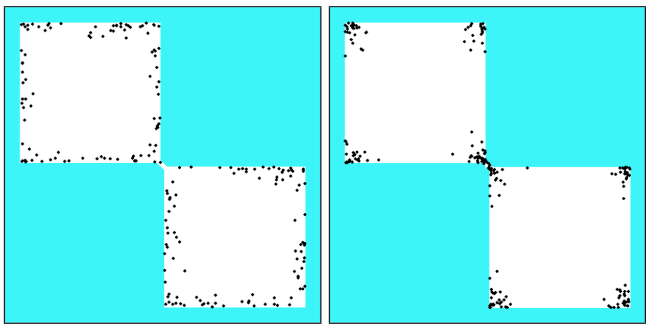


FIG. 1.3 – *Comparaison entre la méthode de tirage gaussien et la méthode basée sur les "bridge tests" (tiré de [Hsu 03]).*

Un autre type d'approche consiste au contraire à chercher les configurations les plus lointaines possible de $\mathcal{C}_{obst}$ pour obtenir le maximum de "visibilité" possible de l'espace libre [Wilmarth 99, Lien 03]. Ces méthodes visent à concentrer les échantillons autour du *diagramme de Voronoï* de $\mathcal{C}_{free}$, également appelé axe médian. Ce processus est réalisé en se basant sur la distance aux obstacles, plus coûteuse à réaliser que la simple détection de collisions.

Contrôle de l'algorithme

Une autre difficulté propre aux réseaux probabilistes est la détermination d'un critère d'arrêt pertinent. L'algorithme *Visib-PRM* [Nissoux 99, Siméon 00] permet d'obtenir un critère d'arrêt en fonction de la portion de $\mathcal{C}_{free}$ couverte par le réseau (c.f. figure 1.4). Le principe de l'algorithme est d'ajouter un nouveau nœud seulement quand la configuration associée sert à relier deux composantes connexes différentes ou quand la configuration n'est pas "visible" (i.e. il n'existe pas de chemin local la reliant à un nœud existant). Le nombre d'échecs à l'insertion d'un nœud est en relation directe avec la couverture du réseau vis à vis de $\mathcal{C}_{free}$. Ainsi, une limitation du nombre maximal d'échecs sert de critère d'arrêt naturel pour ce planificateur. Un autre avantage de cette technique est que la connexité de l'espace libre est capturée au moyen d'une structure de données compacte impliquant une quantité de nœuds et d'arêtes généralement nettement moins grande que pour les autres méthodes.

Détection des collisions

Des algorithmes de détection de collisions évolués [Jiménez 98, Lin 03], ont été développés afin de répondre efficacement aux nombreux tests de collisions effectués par ces méthodes. Ces techniques reposent généralement sur des algorithmes de décompositions hiérarchiques

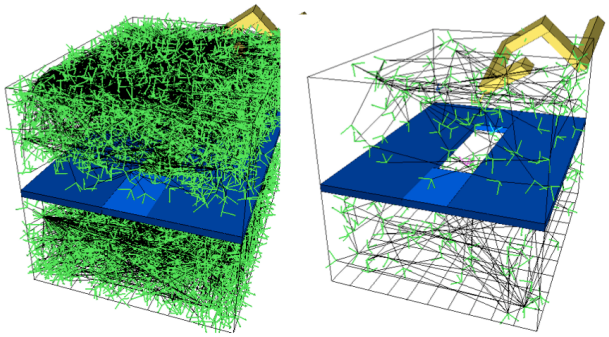


FIG. 1.4 – *Comparaison du nombre de nœuds du réseau entre une méthode PRM (gauche) et une méthode Visib-PRM (droite) pour un environnement de type casse tête 3D (d'après [Nissoux 99]).*

qui précalculent pour chaque objet de l'environnement une approximation multi-niveau de volumes englobants à l'aide de volumes élémentaires tels que des sphères, des boîtes alignées par rapports aux axes du repère de référence, ou encore des boîtes orientées. Malgré la performance de ces détecteurs, les tests de collisions représentent en général la grande majorité des opérations effectuées (en moyenne 90% du temps total de calcul pour les algorithmes PRM standards [van Geem 01]). Partant de cette constatation, la méthode *Lazy-PRM* [Bohlin 00] propose de construire un premier réseau "paresseux" sans considérer les obstacles. Les tests de collisions avec l'environnement ne sont effectués que lors de la phase de requête. Durant cette phase, des chemins du réseau sont itérativement extraits et testés dichotomiquement vis à vis des collisions, et ce jusqu'à ce qu'un chemin entièrement valide soit extrait ou que tous les chemins du réseau aient été testés. La méthode des *Fuzzy-PRM* [Nielsen 00] utilise un principe assez similaire : les nœuds sont testés lors de leur insertion mais la validité des arêtes n'est vérifiée que lors de la phase de requête. De plus, les meilleurs chemins potentiels sont estimés à partir de probabilités de validité associées aux arêtes.

Echantillonnage aléatoire v.s. déterministe

Les algorithmes probabilistes s'appuient sur un échantillonnage de configurations de $\mathcal{C}$. Or il existe une grande variété de méthodes pour échantillonner ces configurations. Une propriété importante est que l'échantillonnage converge de "façon dense" vis à vis de l'espace des configurations. Cela signifie que pour toute configuration q de $\mathcal{C}$ et pour n'importe quelle distance d non nulle, on peut trouver un élément de la séquence d'échantillons à une distance de q inférieure à d. Un tirage (pseudo) aléatoire de chacun des paramètres de configuration est un moyen simple de créer une séquence dense dans $\mathcal{C}$ et reste encore largement utilisé en planification de mouvement.

D'autres critères peuvent cependant être définis pour caractériser ces séquences de points

[LaValle 05]. La *dispersion* qui correspond au rayon de la plus grande boule de $\mathcal{C}$ ne contenant pas d'échantillon[2] permet de caractériser le degré d'uniformité de la répartition des points dans l'espace. Les techniques à base de *grilles* permettent d'obtenir des séquences à dispersion minimale. Un autre type de structure possible est ce qu'on appelle les *lattices*. Cette structure proche des grilles, se différentie par ses vecteurs générateurs qui peuvent ne pas être orthogonaux. En plus de posséder une bonne dispersion, l'avantage de ces structures est qu'à partir d'un point, les plus proches voisins peuvent être obtenus aisément à partir d'opérations élémentaires sur les vecteurs générateurs. On peut également chercher à minimiser la *discrépance* de l'échantillonnage, c'est à dire l'uniformité de la séquence tirée. Les méthodes d'échantillonnage de Halton et de Hammersley permettent d'obtenir une discrépance optimale.

Guidage de l'échantillonnage

Des méthodes récentes proposent pour guider l'échantillonnage, d'exploiter davantage l'information acquise au cours du processus de construction. Ainsi dans [Burns 03], un modèle statistique de l'espace des configurations est créé. L'idée est de tirer les configurations dans les régions où elles augmentent l'information sur le modèle. Ces régions correspondent typiquement à celles pour lesquelles, dans un voisinage donné, certaines configurations testées appartiennent à $\mathcal{C}_{free}$ et d'autre à $\mathcal{C}_{obst}$. Une extension de la méthode à des problèmes de type simple requête est présentée dans [Burns 05b]. De façon similaire, on construit dans [Burns 05a] un modèle représentatif de $\mathcal{C}$, puis on cherche au cours de l'échantillonnage à diminuer au plus vite la variance liée à ce modèle. Enfin dans [Burns 05c], le planificateur tient également compte de la requête courante pour estimer les configurations les plus utiles à échantillonner.

Stratégies de connexion

Différentes stratégies de connexion entre nœuds du graphe peuvent être employées. On peut par exemple pour des raisons d'efficacité, ne chercher à se connecter qu'à un seul nœud pour chaque composante connexe. Cette stratégie aboutit à la construction de graphes à structure d'arbre (e.g. [Nissoux 99]). A l'opposé, on peut choisir de se connecter à l'ensemble des nœuds déjà existants dans le graphe. Cette solution risque d'aboutir à un réseau extrêmement redondant, comportant de nombreuses arêtes. Entre ces deux solutions extrêmes, une solution intermédiaire est de limiter les connexions aux k nœuds les plus proches [Kavraki 96], ou encore aux nœuds situés à une distance inférieure à un seuil donné [Bohlin 00].

Une méthode originale proposée dans [Huang 04], a pour principe de ne tenter une connexion entre deux nœuds que s'ils sont adjacents vis à vis d'une triangulation de Delaunay. Cette méthode a pour but d'augmenter la connexité du graphe dans le cas de problèmes pour lesquels certaines directions sont particulièrement contraintes. Comme il est coûteux de calculer une représentation explicite d'une triangulation de Delaunay, une approximation basée sur

[2] La mesure de la dispersion dépend donc de la métrique utilisée.

des statistiques expérimentales est proposée, ce qui fait malheureusement perdre une partie de l'intérêt de la méthode.

Dans [Nieuwenhuisen 04b], on tente de connecter deux nœuds, seulement si la distance qui les relie dans le graphe est k fois plus grande que la distance les séparant réellement. On dit alors que l'arête formée est k-utile. Cette méthode améliore en moyenne la connexité du réseau, à nombre d'arête équivalent. De plus, elle garantit pour un chemin solution extrait du réseau d'être d'une longueur au pire k fois plus grande que celle du chemin optimal. L'inconvénient, comme pour les méthodes précédentes, est la dépendance vis à vis d'un paramètre qui contrôle le degré de connexité du réseau final. Au chapitre 4, nous présenterons une nouvelle méthode qui permet la construction de réseaux cycliques de faible taille permettant de capturer les différents types de chemins libres de l'espace, sans avoir à introduire de paramètre de contrôle.

Qualité du réseau

Les méthodes probabilistes construisent généralement un chemin solution de mauvaise qualité, qui peut être inutilement long ou passer trop près des obstacles. Une technique simple pour réduire la taille des trajectoire est de découper aléatoirement le chemin en trois parties et d'essayer de remplacer chacune d'elles par le chemin local reliant leurs extrémités. Une autre stratégie [Geraerts 04] cherche à améliorer la qualité des chemins en les éloignant le plus possible des obstacles. La génération de réseaux cycliques de qualité pour des cas simples d'environnements 2D a également été proposé [Nieuwenhuisen 04a].

2.2 Méthodes de diffusion

Contrairement aux méthodes PRM, ces méthodes développent incrémentalement un ou plusieurs arbres qui explorent certaines régions de $\mathcal{C}_{free}$ en vue de résoudre un problème donné. On distingue généralement les versions monodirectionnelles qui ne développent qu'un seul arbre des versions bidirectionnelles qui utilisent les deux configurations données en entrée du problème. L'exploration étant particulièrement efficace dans le cas où la configuration "racine" est contrainte, on privilégiera plutôt un planificateur monodirectionnel dans le cas où une seule des deux configurations est fortement contrainte et un planificateur bidirectionnel dans le cas contraire. Avant de présenter la méthode RRT aujourd'hui la plus populaire, nous rappelons le principe de méthodes de ce type introduites auparavant.

Le planificateur RPP (Randomized Path Planner) [Barraquand 91b] est basé sur une fonction de potentiel qui attire le robot vers sa position finale. Le recours à des mouvements aléatoires pour sortir des minima locaux permet au planificateur d'être complet en probabilité. L'un des inconvénients majeurs de cette méthode est qu'elle nécessite l'introduction d'un certain nombre de paramètres (notamment ceux définissant la fonction potentiel). Or le réglage de ces paramètres pour un problème donné peut être critique vis à vis des performances du planificateur.

La méthode du fil d'Ariane [Bessière 93] exploite de manière efficace l'espace des commandes afin d'explorer l'espace des configurations. Cette exploration utilise une fonction *EXPLORE* pour distribuer des balises dans C_{free} de manière à maximiser leurs distances réciproques. Un deuxième algorithme appelé *SEARCH* permet d'accélérer la recherche d'un chemin et d'atteindre la configuration finale. Enfin, des algorithmes génétiques permettent d'optimiser les fonctions des algorithmes *EXPLORE* et *SEARCH*.

La techique proposée dans [Hsu 97] utilise une stratégie de diffusion basée sur la *densité d'échantillonnage*. Pour tout nœud de l'arbre, on associe une densité correspondant au nombre de nœuds situés dans un certain voisinage. Ensuite, le nœud à étendre est choisi aléatoirement avec une probabilité inversement proportionnelle à la densité qui le caractérise. L'idée ici est de privilégier l'expansion vers les régions encore inexplorées de l'espace qui correspondent aux zones contenant le moins de nœuds. En plus de cette stratégie de diffusion, Sánchez et Latombe proposent dans [Sánchez 03] d'utiliser une méthode de tests de collisions paresseux. Cette approche reprend l'idée des tests de collisions paresseux proposée dans les *lazy-PRM* [Bohlin 00]. L'algorithme construit incrémentalement deux arbres initialisés par les deux configurations données en entrée du problème, mais ne teste pas la validité des arêtes. Ce n'est que lorsque ces deux arbres se rencontrent qu'un chemin potentiellement solution est extrait du graphe, pour être testé de manière dichotomique. Si une collision est détectée, la portion de chemin concernée est retirée du graphe et le processus de construction se poursuit jusqu'à ce que les deux arbres se rejoignent de nouveau.

La méthode d'exploration RRT (*Rapidly-exploring Random Tree*) [LaValle 98], est une méthode extrêmement performante utilisée aujourd'hui dans de nombreuses applications. Cette méthode a été initialement proposée pour traiter le cas des problèmes non-holonomes ou kinodynamiques. Par la suite, elle a également été adaptée au cas plus simple des systèmes holonomes qui peuvent être directement formulés dans $\mathcal{C}$ [Kuffner 00, LaValle 01a]. L'intérêt de cet algorithme vient d'un guidage efficace de l'exploration : la probabilité pour un nœud d'être sélectionné pour l'extension est proportionnelle à sa région de Voronoï. Une description plus détaillée de cet algorithme et une analyse de ses performances sera proposée en début de chapitre 3, avant la présentation d'un nouvel algorithme de la même famille.

2.3 Extensions du problème

De nombreuses recherches portent sur l'extension des méthodes probabilistes à des problèmes plus complexes que le problème initial de la planification de mouvement (c.f. exemple figure 1.5). Nous nous proposons ici de présenter certaines d'entre elles.

Contraintes de chaîne fermée

Les contraintes liées à la fermeture de chaînes cinématiques est un cas particulier de contraintes holonomes. Contrairement au cas des chaînes ouvertes, l'espace des configurations

FIG. 1.5 – *Une version "moderne" du problème du déménageur de piano. Le système mis en œuvre incorpore, en plus du piano, un personnage virtuel et deux manipulateurs mobiles. Ce système à nombreux degrés de liberté doit naviguer dans un environnement encombré en intégrant en plus des contraintes de fermeture cinématique (d'après [Esteves 06]).*

vérifiant les contraintes de chaînes fermées est une variété de $\mathcal{C}$, ce qui rend le problème plus difficile. Pour une formalisation plus détaillée du problème et une présentation des algorithmes on pourra se référer à [Basu 00, LaValle 99, Han 01, Cortés 03].

Contraintes de manipulation

Le problème de la manipulation d'objets peut être vu comme une instance plus compliquée du problème de la planification de mouvement [Alami 89, Siméon 03]. Dans sa version la plus simple, ce problème met en jeu un robot, un objet manipulé qui correspond d'après nos définitions à un obstacle mobile, et des obstacles statiques qui représentent l'environnement. Les opérations de manipulation se composent en général de deux types de mouvements séparés par des prises/poses des objets : les mouvements de *transfert* pendant lesquels l'obstacle mobile est manipulé par le robot et les mouvements de *transit* pendant lesquels seul le robot bouge (les objets mobiles étant alors positionnés à un placement stable).

Contraintes non-holonomes

Les systèmes non holonomes ont suscité de nombreux travaux en planification de mouvement [Laumond 86, Laumond 98a]. Ces systèmes sont caractérisés par des contraintes non intégrables qui lient les paramètres de configuration. Plusieurs travaux portent en particulier sur l'étude des systèmes contrôlables en temps petit [Laumond 98b, Laumond 01, Lamiraux 01b] pour lesquels, tout chemin de $\mathcal{C}_{free}$ peut être approximé par une séquence finie de chemins respectant les contraintes de non-holonomie. Des planificateurs probabilistes ont également été construits pour des robots de type voiture [Svestka 97b] ou des systèmes à remorques [Sekhavat 98, Lamiraux 97].

Contraintes kinodynamiques

Le terme de planification “kinodynamique” [Donald 93] fait référence aux problèmes pour lesquels des bornes sur les vitesses et les accélérations doivent être satisfaites. Plus généralement, on considère maintenant comme “kinodynamique” un problème impliquant des contraintes du deuxième ordre (ou plus) sur les paramètres de configuration. Notons que dans la majeure partie des cas, ces contraintes ne représentent pas un type particulier de contraintes holonomes. On peut cependant trouver des systèmes combinant ces deux types de contraintes [Fraichard 99].

Planification multi-robots

Deux grandes familles d’approches existent pour planifier le mouvement simultané de plusieurs robots. D’une part les approches centralisées [Schwartz 83c] qui se placent dans l’espace des configurations produit de l’espace des configurations de chacun des robots et d’autre part, les approches distribuées [Kant 86] qui planifient indépendamment les trajectoires de chacun des robots et cherchent ensuite à les coordonner. Les approches centralisées sont généralement complètes et garantissent de trouver une solution. En pratique leur complexité [Hopcroft 84] les limite cependant à des problèmes impliquant peu de robots. Au contraire, bien qu’incomplètes, les approches distribuées permettent d’obtenir des solutions de manière efficace quand le problème n’est pas trop contraint. Parmi ces méthodes, les approches prioritaires [Erdmann 87] définissent un ordre de priorité entre les robots et calculent de manière consécutive des chemins valides pour les robots en commençant par ceux ayant la plus grande priorité. Ces calculs se font en se plaçant dans l’espace des configurations$\times$temps (c.f. section 3.1) et en prenant pour obstacles les chemins des robots de plus grande priorité. Dans [Warren 90], cette approche est combinée à des champs de potentiel pour résoudre les éventuels conflits locaux. La notion de diagramme de coordination [O’Donnel 89] est également utilisée pour représenter pour deux robots, les positions respectives le long des chemins qui mènent à une collision mutuelle. Dans [Siméon 02], une extension permet d’utiliser le diagramme de coordination pour résoudre des problèmes impliquant plus d’une centaine de robots.

Robots déformables

D’autres travaux portent sur l’extension des méthodes aléatoires au cas de robots représentés par des objets déformables [Holleman 98]. La principale difficulté de cette extension est que l’espace des configurations d’un objet déformable est potentiellement de dimension infinie. Le formalisme introduit dans [Lamiraux 01a], permet de prendre en compte des propriétés physiques telles que l’élasticité dans la résolution des problèmes. Un exemple d’application pour un robot de type plaque élastique est présenté. Récemment, une nouvelle méthode de planification pour robots déformables en environnement complexe a été présentée [Gayle 05]. Cette méthode permet de construire des chemins en considérant en plus des contraintes géométriques liées aux obstacles, des contraintes physiques telles que la préservation du volume, le contrôle de la tension de surface ou la minimisation d’un critère énergétique.

Notons enfin que d'autres contraintes affectent le mouvement dans le monde réel, certaines étant plus dures à traduire en terme d'équations mathématiques. Par exemple, dans le cadre de problèmes impliquant des *personnages virtuels*, il est possible en vue d'un résultat réaliste, de faire appel à d'autres techniques issues de l'animation graphique telles que la capture de mouvement sur personnage réel [Kuffner 99, Pettre 02].

3 Planification et scènes dynamiques

Les algorithmes de planification de mouvements considèrent dans la plupart des cas des environnements statiques parfaitement connus. Dans de nombreuses situations, des modifications de la scène nécessitent pourtant d'être prises en compte dans le calcul du mouvement. On parle de façon générale de problèmes à *environnement dynamique*. Les travaux concernant les environnements dynamiques s'articulent plus particulièrement autour de trois grandes problématiques. La première concerne les situations pour lesquelles l'évolution dans le temps de la position des obstacles mobiles est supposée connue. Dans ce cas, la prise en compte de ces obstacles peut être faite dès la phase de planification. La deuxième problématique porte sur la navigation des robots, qui nécessite de conjuguer des méthodes de planification avec des techniques d'exécution de trajectoire. La dernière problématique concerne directement le sujet de cette thèse. L'objectif est de développer des méthodes de planification plus performantes pour rendre compte des différents changements susceptibles d'avoir lieu dans l'environnement. Les trois sous-sections suivantes présentent de façon plus développée ces trois problématiques.

3.1 Planification et prise en compte du temps

Dans le cas d'un environnement 3D contenant plusieurs obstacles mobiles qui suivent des trajectoires connues, on sait que le problème est *PSPACE*-difficile si la vitesse maximale est bornée, et NP-difficile dans le cas contraire [Reif 85]. Le planificateur doit alors calculer une trajectoire paramétrée en temps, au lieu d'un simple chemin géométrique.

Dans [Kant 86], une approche en deux phases est proposée. D'abord, les obstacles mobiles sont ignorés et un chemin valide vis à vis des obstacles statiques est calculé. Dans un deuxième temps, la vitesse du robot le long du chemin est contrôlée pour éviter les collisions avec les obstacles. Le planificateur donne de bons résultats quand le nombre d'obstacles mobiles est faible et/ou l'environnement n'est pas très encombré. Pour augmenter les potentialités du planificateur [Fujimua 95], on propose d'étendre la méthode en générant un réseau de chemins.

La notion d'espace des vitesses interdites noté $\mathcal{V}_{obst}$ est introduite dans [Shiller 96]. D'un point de vue général, $\mathcal{V}_{obst}$ est à l'espace des vitesses ce que $\mathcal{C}_{Obst}$ est à l'espace des configurations : il désigne l'ensemble des vitesses instantanées du robot qui entraînent une collision dans un temps borné appelé *horizon de temps*. Ce concept est utilisé pour générer, pour le robot, des trajectoires ayant un profil de vitesses n'appartenant pas à $\mathcal{V}_{obst}$, dans le cas où les obstacles

se déplacent en ligne droite et à vitesse constante. Récemment, [Large 03], présente une expression analytique des bords de $\mathcal{V}_{obst}$, qui permet d'étendre la méthode à des trajectoires quelconques des obstacles et de remplacer l'information booléenne de collision ou non-collision par le temps de collision associé à chaque vitesse.

La notion d'espace des configurations$\times$temps est introduit dans [Erdmann 87]. Cet espace correspond à $\mathcal{C}$ augmenté d'une dimension qui représente le temps. Une extension à des contraintes kinodynamiques [Fraichard 93] introduit la notion d'espace des états$\times$temps, où l'état d'un robot est défini par l'union de ses paramètres de configuration et de ses dérivées (premières et éventuellement secondes) par rapport au temps. L'utilisation des états$\times$temps a également été reprise dans [Hsu 02]. Le planificateur diffuse dans cet espace un arbre de recherche en vue de connecter les points états-temps donnés en entrée du problème. Une analyse algorithmique montre notamment la convergence en probabilité de la méthode suivant certaines hypothèses propres à la géométrie de l'espace des états-temps. Enfin, le planificateur est utilisé pour replanifier en ligne la trajectoire d'un robot circulaire entouré de plusieurs obstacles mobiles.

Une autre approche proposée dans [van den Berg 04] utilise un algorithme en deux étapes. Un premier réseau à base de cycles est calculé sans prendre en compte les obstacles dynamiques. Ensuite, à partir de ce réseau, une trajectoire valide est calculée pour une requête donnée. La trajectoire est déterminée en diffusant les contraintes temporelles le long du réseau et en recherchant un plus cours chemin sur une grille dans l'espace des états-temps. Une extension à des problèmes multi-robots a également été proposée dans [van den Berg 05b].

3.2 Planification et exécution

Dans de nombreux travaux appliqués à la navigation de robots mobiles, les méthodes de planification sont combinées à des méthodes d'évitement d'obstacles, telles que la méthode des potentiels [Khatib 86]. Ces méthodes, généralement basées sur un raisonnement local, ont avant tout pour but d'assurer la sécurité du robot malgré les aléas liés à l'exécution du mouvement. Bien que très réactives, elles possèdent cependant un inconvénient majeur : elles sont souvent sujettes au problème des minima locaux et peuvent conduire à des situations de blocage dans lesquelles le robot sera incapable d'atteindre sa positon but. De nombreuses méthodes d'évitement d'obstacles existent dans la littérature et on laissera le lecteur se référer aux articles de référence. Citons simplement la méthode des *Vector Field Histogramm* [Borenstein 91, Ulrich 98, Ulrich 00], les techniques de *Curvature Velocity* [Simmons 96, Ko 98], les *Dynamic Window* [Fox 97, Brock 99], ou encore l'approche basée sur les *Nearness Diagram* [Minguez 00].

Une autre approche établit le lien entre planification et exécution. Il s'agit de la technique de la *bande élastique* [Quinlan 93]. Le principe est de maintenir en permanence un passage sans collision (appelé bande) entre le robot et son but. Ce passage est représenté au moyen d'un ensemble de disques (2D) ou de boules (3D) de l'espace libre, appelés *bulles*. Le diamètre de ces

bulles peut s'accroître ou se réduire pour couvrir au mieux l'espace libre, en fonction des changements de l'environnement. Leur nombre peut également varier pour éviter toute interruption de la bande. Deux types de forces entrent en action pour maintenir leur cohérence : des forces attractives entre bulles consécutives afin de maintenir la bande "tendue" et des forces répulsives avec les obstacles pour maintenir la bande éloignée de ces derniers. L'équilibre de ces forces tend vers un chemin sûr jusqu'au but. La mise à jour des forces assure l'adaptation de ce chemin en temps réel en fonction des obstacles. L'approche, initialement développée pour planifier des chemins géométriques en temps réel [Quinlan 93], a été étendue ensuite dans [Khatib 97] à des robots non-holonomes utilisant une méthode locale de type Reed et Shepp. Pour gérer les "cassures" de la bande, nécessaires dans le cas d'un obstacle passant orthogonalement à la trajectoire et pour éviter à la bande de se faire coincer, une extension à été proposée sous le terme de bandelette élastique [Brock 00]. L'idée est de relâcher temporairement la contrainte entre deux bulles adjacentes pour laisser passer un obstacle mobile. De plus, pour limiter les processus de replanification complète, un ensemble de bandes de secours est conservé, servant de solutions alternatives rapidement disponibles. Dans [Lamiraux 04], une méthode générique de déformation réactive de trajectoire pour robot non-holonome est proposée, en se basant sur un mécanisme de perturbation des commandes d'entrée du système robotique utilisé.

Les *méthodes hybrides*, combinent au sein d'un même algorithme des stratégies de planification de trajectoire et des méthodes d'évitement d'obstacle. Certaines de ces méthodes correspondent à une extension des méthodes d'évitement d'obstacles en une version globale. Ainsi dans [Brock 99], on propose de généraliser la technique des Dynamic Window, en la couplant à des méthodes standards de planification. De la même façon, l'extension des Nearness Diagram en méthode globale est proposée dans [Minguez 02]. Dans [Large 04], on utilise la représentation des trajectoires des obstacles dans l'espace des vitesses instantanées du robot, pour éviter les obstacles dynamiques. Cette stratégie est ensuite combinée à des techniques d'expansion de graphes et des techniques de prédiction des mouvements des obstacles, permettant au final de construire un planificateur global incrémental. D'autres méthodes se basent au contraire sur des techniques de planification et les adaptent pour contrôler l'exécution d'une trajectoire. C'est le cas par exemple des travaux présentés dans [Hsu 02, Bruce 02]. Dans [Petti 05], on cherche à construire des mouvements sûrs, en planifiant de manière itérative tout en s'assurant à chaque intervalle de temps que l'on n'atteint pas des états du système pour lesquels une collision serait inévitable. Une autre méthode [Brock 01] tente d'exploiter l'information de connexité liée à $\mathcal{W}$ pour guider la recherche dans $\mathcal{C}$. L'algorithme se décompose en deux étapes. La première consiste à calculer un volume libre de $\mathcal{W}$ contenant au moins un chemin solution. Ce calcul se fait à l'aide d'un algorithme de propagation par vagues de boules libres de $\mathcal{W}$ à partir de la position initiale. Dans un second temps, un chemin solution est calculé à l'aide de techniques de champs de potentiel puis une fonction de navigation qui s'appuie sur le tunnel précédemment calculé adapte la trajectoire en fonction des obstacles mobiles.

En général, les stratégies couplant planification et exécution fonctionnent bien pour les problèmes où un chemin solution suffisamment dégagé peut facilement être déterminé (i.e. suffisamment loin des obstacles). Elles supposent de plus, une certaine similarité entre la connexité de $\mathcal{W}$ et celle de $\mathcal{C}$. Pour ces raisons, elles ne sont ni adaptées aux systèmes mécaniques complexes, ni aux environnements encombrés. Le cadre de travail dans lequel s'inscrit cette thèse concerne au contraire des problèmes dont la dynamique est plus faible, mais où les systèmes mis en œuvre sont plus complexes.

3.3 Planification et scènes dynamiques

Pour tenir compte des changements de position des objets dans les scènes dynamiques, les méthodes de planification à base de graphes doivent comporter une phase de mise à jour, pendant laquelle la validité des éléments du graphe est testée. Compte tenu des contraintes temporelles généralement fortes pour les problèmes dynamiques, ces méthodes doivent être particulièrement efficaces. Une première idée est d'utiliser des techniques de *mapping* entre l'environnement $\mathcal{W}$ et l'espace des configurations $\mathcal{C}$. Ces techniques construisent une structure de données qui fait correspondre des cellules élémentaires de l'environnement (pixels en 2D, voxels en 3D) aux différents nœuds et arêtes du graphe. Cette construction se fait en balayant l'espace de travail à partir d'un obstacle élémentaire de la taille des cellules et en testant pour chaque position prise par cet obstacle dans l'environnement, quelles sont les portions du réseau affectées. Lors de la mise à jour du graphe, les cellules de l'environnement invalidées permettent de déterminer sans avoir à réaliser de nouveaux tests de collisions, quels sont les nœuds et arêtes affectés. Une méthode "primitive" de mapping est présentée dans [Lean 96a]. Elle construit une structure qui mémorise les portions du graphe invalides pour les différents placements potentiels d'un obstacle qui peut apparaître ou disparaître dans la scène. C'est [Leven 00] qui propose véritablement la première représentation de l'espace à travers une structure de données et qui l'applique à des robots manipulateurs plans. Une limitation de la méthode est qu'elle nécessite une importante taille mémoire pour stocker l'information de mapping. Pour essayer de réduire la taille mémoire nécessaire, une méthode de compression de données basée sur la cohérence spatiale est proposée dans [Leven 02]. Une autre limitation est la décomposition de l'espace en cellules qui rend la méthode complète uniquement par rapport à la résolution donnée.

Une autre idée consiste à utiliser le principe des réseaux paresseux [Bohlin 00, Sánchez 03, Nielsen 00] également utilisés pour les problèmes en environnement statique (c.f. section 2.1). Pour ces réseaux, le test de validité des arêtes et éventuellement des nœuds n'est réalisé qu'au moment de la phase de requête. En environnement dynamique, ces tests se font donc directement vis à vis du contexte courant. Ainsi cette stratégie permet d'éviter des mises à jour inutiles pour des contextes différents du contexte de requête. De plus, la méthode utilisée se base sur une mise à jour du réseau seulemement partielle, ce qui permet de rendre l'approche intéressante dans les problèmes de scènes dynamiques où les contraintes d'efficacité sont fortes. Contrairement aux méthodes de mise à jour paresseuse que nous présenterons chapitre 2, ces

méthodes ne distinguent pas la scène statique des obstacles mobiles, et n'exploitent donc pas le caractère invariant de la partie statique.

A notre connaissance, seul [Lean 96b] propose d'utiliser une méthode de réparation des portions invalides, à la suite d'une étape de mise à jour. Cette méthode répare les chemins du graphe initial en projetant hors des obstacles des configurations invalidées à l'aide d'un mouvement brownien biaisé vers les régions extérieures. Cependant, cette méthode "ad hoc" utilisée pour résoudre un problème spécifique en milieu industriel se limite à des robots rigides simples.

La structure des graphes utilisés dans les problèmes de planification en environnement dynamique est également très importante. Dans la littérature, cette question est pourtant très peu considérée. Les réseaux généralement utilisés [Leven 00, Bohlin 00, Lean 96b, Lean 96a] sont construits à partir des stratégies traditionnelles de connexion (k nœuds les plus proches, distance maximale) utilisées pour construire des réseaux en environnement statique (c.f. section 2.1). La structure finale est généralement dense et très redondante, ce qui permet de capturer les différents chemins de l'espace libre mais engendre des coûts de mise à jour plus élevés. Le critère d'ε-robustesse [Leven 00] a été proposé pour estimer la robustesse des réseaux vis à vis des obstacles mobiles : un réseau est ε-robuste si aucun obstacle de rayon égal ou inférieur à ε ne peut en casser la connexité. En pratique, l'évaluation de ce paramètre est cependant difficile.

Chapitre 2

Planificateur pour Environnements Changeants

Dans ce chapitre, nous présentons une extension des méthodes de planification par réseaux probabilistes afin de traiter efficacement des scènes dynamiques composées à la fois d'obstacles statiques et mobiles. La méthode proposée repose sur la combinaison des méthodes PRM avec des méthodes de diffusion afin d'exploiter leurs avantages respectifs. Elle intègre également une mise à jour efficace du réseau par des mécanismes de mémorisation et d'évaluation paresseuse. Les résultats présentés en fin de chapitre montrent que la performance de l'approche permet son utilisation pour la déformation "temps-réel" de trajectoires en présence d'obstacles mobiles perturbant l'exécution du mouvement planifié.

1 Exemple introductif

La figure 2.1 présente un exemple d'environnement changeant. Le robot (une plate-forme mobile transportant une planche), évolue au sein d'un environnement d'intérieur de type bureau. En plus des deux portes alternativement ouvertes ou fermées, la scène comporte trois autres obstacles mobiles qui peuvent apparaître autour de la table (c.f. image en bas à droite). La figure montre également pour une même requête mais pour trois contextes différents, trois chemins calculés en temps-réel par notre planificateur.

Cet exemple introductif illustre les difficultés liées à la prise en compte d'obstacles mobiles par les algorithmes de planification. On doit d'une part considérer plusieurs types d'obstacles : alors que certains peuvent avoir un nombre discret de placements (typiquement les portes d'un environnement), d'autres peuvent avoir un ensemble infini de placements possibles. Certains obstacles peuvent aussi apparaître, disparaître, ou changer de position au cours des requêtes

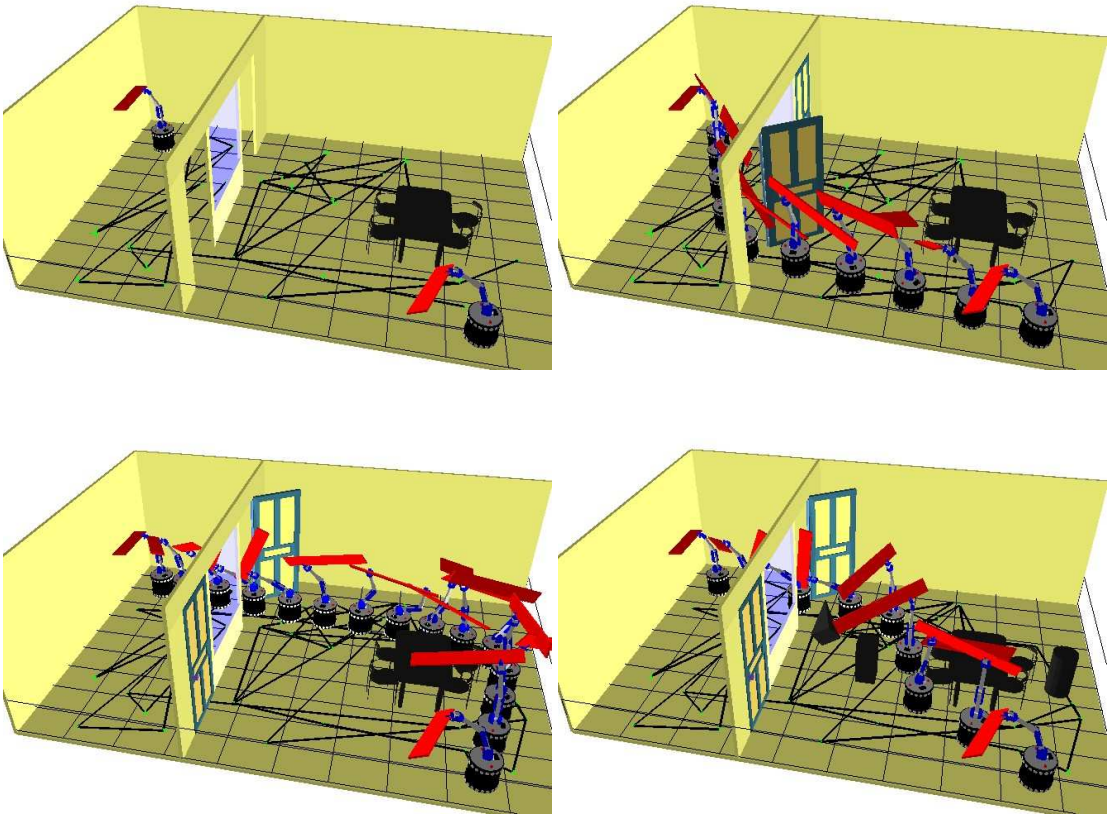


FIG. 2.1 – *Configurations initiales et finales d'une plate forme manipulatrice et trois chemins solution obtenus pour trois contextes de scène différents*

successives. D'autre part, un obstacle mobile changeant localement de position peut conduire à une modification globale des chemins solutions.

Les approches probabilistes sont aujourd'hui les seules méthodes capables de traiter de manière générique des problèmes pour des systèmes robotiques complexes. En l'état, elles sont cependant mal adaptées aux problèmes dynamiques. D'un coté les algorithmes de diffusion qui construisent un nouveau graphe à chaque requête, perdent à chaque fois toute information liée aux obstacles statiques. De l'autre, les méthodes à requêtes multiples construisent des réseaux qui peuvent être invalidés par les obstacles mobiles. L'utilisation de ces réseaux ne peut donc se faire sans l'ajout d'un mécanisme de mise à jour pour tenir compte du contexte courant.

Nous proposons ici un planificateur hybride, qui exploite des avantages liés à ces deux familles de méthodes : la construction de réseaux probabilistes permet de capturer l'espace libre privé d'obstacle mobile et une méthode de diffusion permet ensuite de reconnecter les portions

invalides pour un contexte donné. On utilise également une stratégie efficace qui combine mise à jour du réseau et recherche d'un chemin solution. L'objectif de cette stratégie est de réduire au maximum les tests de validité qui représentent généralement les opérations les plus coûteuses.

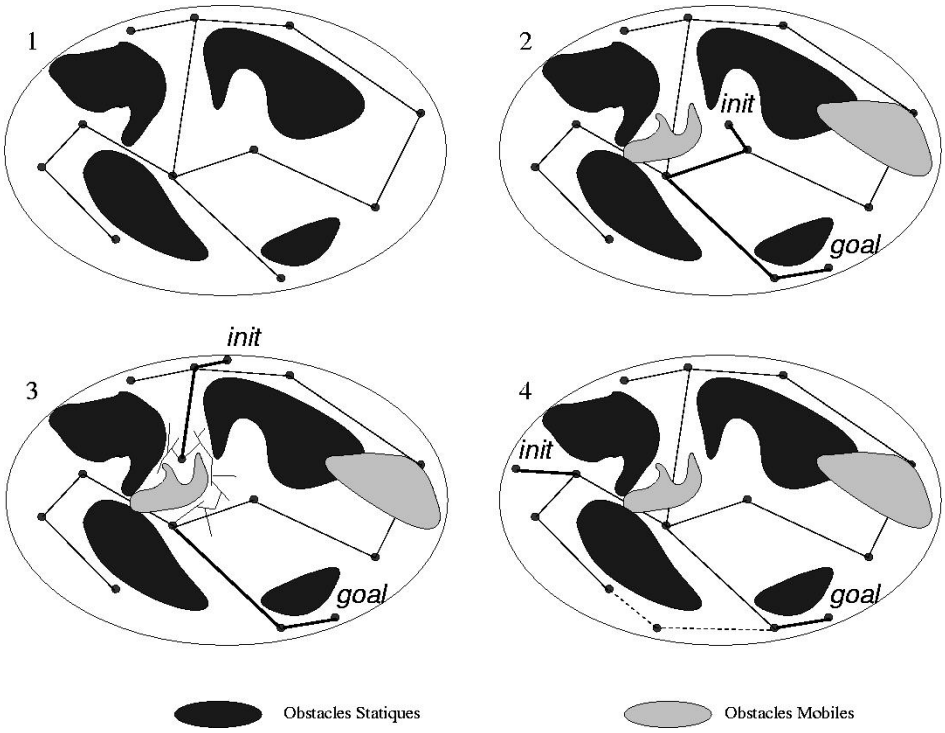


FIG. 2.2 – *Un réseau valide vis à vis de l'environnement statique est d'abord calculé (1). Au cours des requêtes, un chemin solution peut être trouvé directement au sein de ce réseau (2) ou à l'aide d'une technique de connexion des arêtes cassées (3). Si le réseau existant ne permet pas d'obtenir une solution, de nouveaux nœuds sont insérés et le réseau est renforcé à travers l'apparition de cycles (4).*

2 Principe de l'approche

Nous présentons ici le principe général de l'approche décrite dans [Jaillet 04]. Elle comprend deux étapes principales. La première est une étape d'initialisation, pendant laquelle est construit un réseau valide dans l'espace des configurations, privé d'obstacles mobiles (figure 2.2-1). Comme cette étape n'est effectuée qu'une fois pour un environnement donné, les contraintes au niveau du temps de construction sont assez faibles. Le but de cette étape est de construire une structure de données qui permette par la suite de traiter plus efficacement les requêtes de planification. Au cours d'une requête, les nœuds extrémaux sont connectés au

réseau, puis les portions sont mises à jour par des tests de collisions limités aux obstacles mobiles. Les arêtes invalides sont alors étiquetées comme bloquées dans le réseau. Dans certains cas, cet étiquetage est suffisant pour obtenir un chemin solution sans calcul supplémentaire (figure 2.2-2). Sinon, un mécanisme de reconnexion locale essaye de réparer les arêtes invalides (figure 2.2-3). Ce mécanisme permet de renforcer la robustesse du réseau vis à vis des obstacles mobiles et d'améliorer la performance. Finalement, si le réseau courant ne permet pas de répondre à la requête, une phase de renforcement permet d'introduire de nouveaux nœuds et arêtes. Ces nouveaux éléments qui peuvent conduire à l'apparition de cycles rendent également le réseau plus robuste (figure 2.2-4).

L'efficacité de l'approche repose aussi sur plusieurs autres méthodes qui permettent d'éviter au mieux les tests de collisions inutiles. En particulier le planificateur inclut les éléments suivants :

– Une méthode *d'évaluation paresseuse* : elle permet de sélectionner les connexions des nœuds de requête les plus prometteuses et de limiter la mise à jour du réseau à certaines portions pertinentes.

– Un *processus de mémorisation* : les tests de collisions réalisés au cours des requêtes successives sont mémorisés pour être à nouveau exploités quand un contexte déjà rencontré se présente.

L'architecture globale du planificateur est présentée figure 2.3.

2.1 Réseau initial

Le réseau calculé lors de l'initialisation doit être uniquement valide vis à vis des obstacles statiques. Ensuite, lors d'une requête de planification, le calcul d'un chemin solution nécessite la mise à jour de certaines portions du réseau précalculé. On associe donc à chacun des nœuds et arêtes du réseau trois états ou *étiquettes* possibles, en fonction du contexte : valide, invalide, ou non testé.

N'importe quelle méthode de construction de réseaux probabilistes peut être à priori utilisée pour la construction (c.f chapitre 1 section 2). Cependant, pour que le planificateur soit performant lors des phases de requête, ces réseaux doivent vérifier deux propriétés à priori antagonistes. D'un coté le réseau doit être de faible densité pour que sa mise à jour ne soit pas pénalisante. De l'autre il doit être suffisamment riche pour contenir des chemins alternatifs lorsque certaines arêtes sont invalidées par les obstacles mobiles.

Le choix de la méthode la mieux adaptée est discuté en fin de chapitre. Par ailleurs, cette question fera l'objet des chapitres 4 et 5 qui décrivent deux méthodes originales de construction de réseaux en adéquation avec ces propriétés : des *réseaux de rétraction* (chapitre 4), qui permettent de capturer dans une structure de données réduite les différents chemins de l'espace difficilement déformables entre eux et des *réseaux robustes* aux changements de position des obstacles mobiles (chapitre 5).

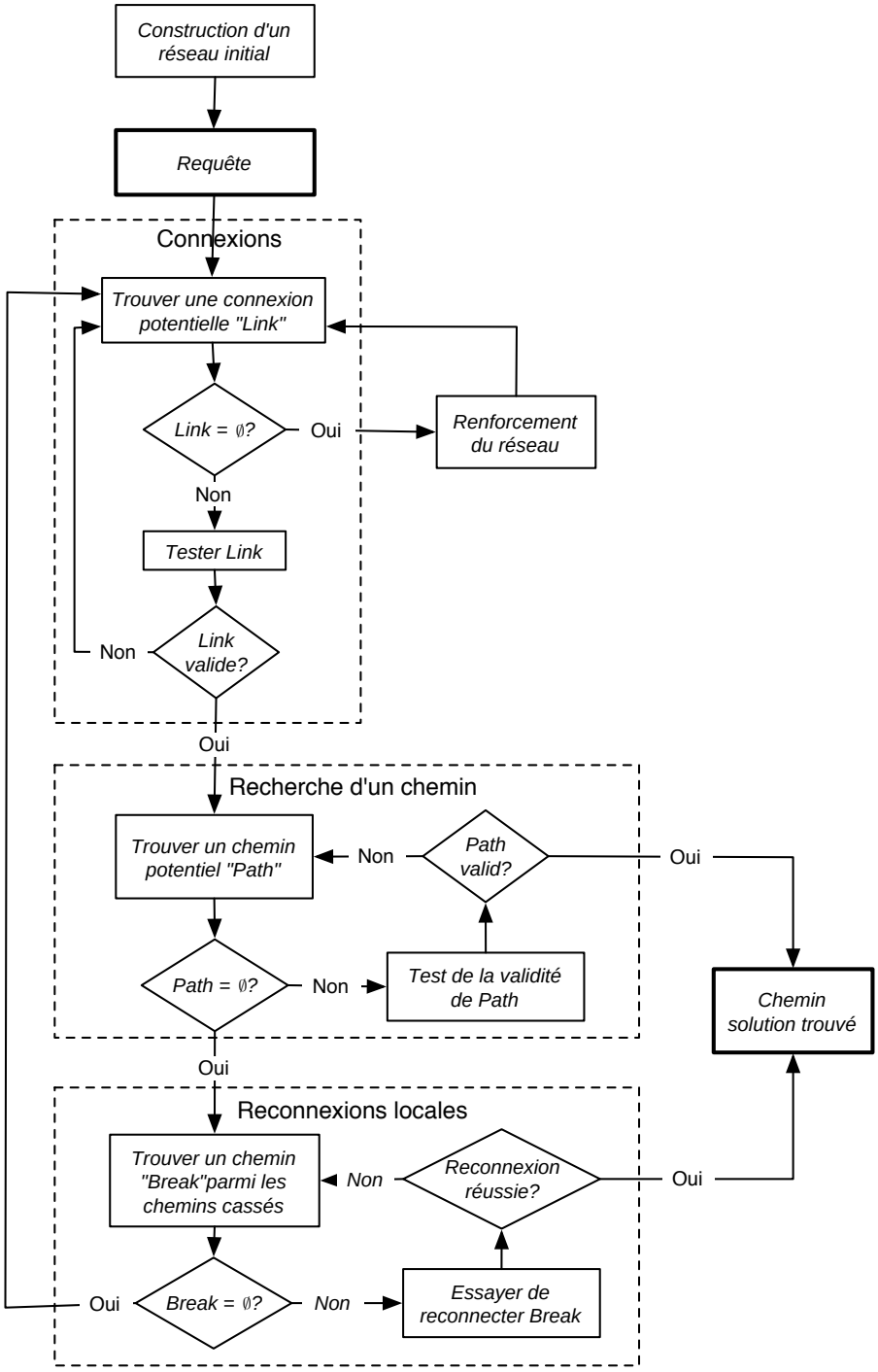


FIG. 2.3 – *Architecture globale du planificateur pour environnements changeants.*

2.2 Requête de planification

La résolution d'une requête de planification est généralement effectuée en trois étapes successives [Lean 96a] :

– Mise à jour de l'ensemble des nœuds et arêtes du réseau initial et calcul des composantes connexes associées au contexte courant.

– Connexions des nœuds de requête aux nouvelles composantes connexes formées.

– Recherche d'un chemin solution dans le réseau ainsi mis à jour.

En pratique, cette méthode "naïve" s'avère peu performante. En effet, le test systématique des arêtes du réseau est dans la plupart des cas inutile, en particulier lorsque le réseau contient un chemin valide vis à vis de l'environnement statique et répondant à la requête de planification (figure 2.4-1). Dans la section suivante, nous proposons une méthode plus performante qui teste en priorité les arêtes associées aux chemins les plus prometteurs. La connexion systématique des nœuds de requête à toutes les composantes est également à éviter (figure 2.4-2 et 2.4-3). Ces opérations sont d'ailleurs d'autant plus pénalisantes qu'ici, les tests de validité font aussi intervenir les obstacles statiques. Notre méthode permettra au contraire de limiter au maximum ces connexions et de sélectionner en priorité celles qui ont le plus de chances d'être valides.

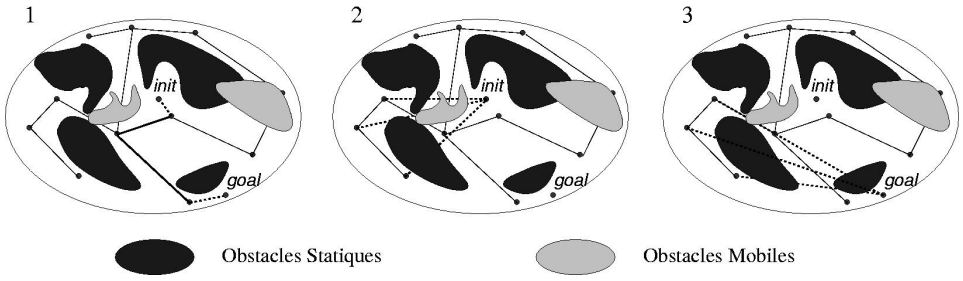


FIG. 2.4 – *Un chemin solution peut souvent être extrait du réseau sans étiqueter l'intégralité des arêtes (1). De la même façon, la connexion des nœuds de requête à l'ensemble des composantes est souvent inutile (2 et 3).*

Pour cela, notre méthode repose sur une stratégie d'évaluation paresseuse qui consiste en une mise à jour graduelle du réseau parallèlement à la recherche d'un chemin solution. C'est sur cette mise en parallèle du processus de recherche et du processus de mise à jour que repose la performance de l'algorithme. De plus, comme mentionné précédemment, la robustesse d'un chemin du réseau est améliorée, grâce à une opération de reconnexion qui répare certaines arêtes invalidées du réseau. Ainsi trois opérations correspondant aux blocs de la figure 2.3 sont réalisées de manière itérative jusqu'à ce qu'un chemin valide soit trouvé :

– La connexion des nœuds de requête aux nœuds du réseau (bloc *Connexion*).

– La recherche d'un chemin solution au sein du réseau (bloc *Recherche d'un chemin*).

– La réparation des chemins invalides à l'aide d'une méthode de reconnexion (bloc *Reconnexions locales*).

Ces trois opérations ainsi que la méthode d'étiquetage paresseuse des arêtes sont détaillées au cours de la section suivante.

3 Mise à jour du réseau lors des requêtes de planification

3.1 Connexions au réseau

Le but de cette étape (bloc *Connexions* figure 2.3) est de sélectionner en fonction du réseau partiellement mis à jour, les nœuds candidats auxquels connecter les nœuds de requête. Ici, on suppose donc que certaines arêtes ont déjà été étiquetées valides ou invalides, alors que d'autres sont encore non testées. Dans ce contexte, on dira qu'un nœud est *potentiellement atteignable* à partir d'un autre nœud s'il existe un chemin qui les relie et qui ne comporte que des arêtes valides ou des arêtes qui n'ont pas encore été testées. Au contraire, un nœud est *non atteignable* à partir d'un autre si tous les chemins qui les relient comportent au moins une arête étiquetée comme bloquée. Cette notion permet de regrouper les différents nœuds du réseau en trois sous-ensembles :

– K_{init}, nœuds potentiellement atteignables à partir du nœud initial.

– K_{goal} nœuds potentiellement atteignables à partir du nœud but.

– K_{na} nœuds qui, pour la mise à jour courante, sont non atteignables à partir des nœuds de requête.

L'utilisation de ces trois sous-ensembles conduit à distinguer trois types de connexions des nœuds de requête parmi lesquelles sera choisie la plus prometteuse. L'algorithme associé à ce mécanisme de sélection est présenté figure 2.5.

Les trois sous ensembles de nœuds (K_{init}, K_{goal} et K_{na}) font partie des données d'entrée de l'algorithme. Au début d'une requête, ces ensembles sont initialisés en considérant que tous les nœuds appartiennent à K_{na}. Ensuite ces ensembles sont mis à jour à la suite des connections de nœuds de requêtes ou des mises à jour de chemins du réseau (fonction *UpdateK* des algorithmes *Connect* et *ExtractPath*). La fonction *BestNode,* permet de déterminer les nœuds les plus proches de ces ensembles pour lesquels des connexions n'ont pas encore été essayées. On envisage alors les trois types de connexions suivantes :

– L_{init}, connexion du nœud initial aux nœuds potentiellement atteignables à partir du nœud final.

– L_{goal}, connexion du nœud final aux nœuds potentiellement atteignables à partir du nœud initial.

– L_{na} connexion simultanée des deux nœuds de requête au troisième sous-ensemble, regroupant les nœuds non atteignables à partir de la mise à jour courante.

Parmi ces trois types de connexions potentielles, celle associée à la plus courte distance est finalement sélectionnée (ligne 10 figure 2.5).

Avec cette méthode, la sélection de la connexion intervient donc à deux niveaux. La fonction *BestNode*, permet de déterminer pour chaque ensemble K_{init}, K_{goal}, K_{na}, les connexions les

Connect($G, K_{init}, K_{goal}, K_{na}$)
1 LinkValid ← False
2 **While** LinkValid = False
3 n_{init} ← BestNode($K_{goal}, init$)
4 L_{init} ← $n_{init} - init$
5 n_{goal} ← BestNode($K_{init}, goal$)
6 L_{goal} ← $n_{goal} - goal$
7 n^{na}_{init} ← BestNode($K_{na}, init$)
8 n^{na}_{goal} ← BestNode($K_{na}, goal$)
9 L_{na} ← $(n^{na}_{init}\text{-}init) \cup (n^{na}_{goal}\text{-}goal)$
10 Link ← BestLink($L_{init}, L_{goal}, L_{na}$)
11 **If** Link = ∅
12 Return ∅
13 **End If**
14 LinkValid ← TestLink(Link)
15 **End While**
16 UpdateK(G)
17 Return Link

FIG. 2.5 – *Algorithme Connect, de recherche des meilleurs types de connexion des nœuds de requête au réseau.*

plus courtes non encore testées. L'algorithme sélectionne ensuite la connexion à tester comme étant la plus courte des trois groupes. La méthode permet donc au final de choisir parmi toutes les connexions présentant un intérêt pour le calcul d'un chemin solution celle étant la plus courte non encore testée qui a le plus de chances d'être valide.

La scénario de la figure 2.6 illustre ce mécanisme de connexion. Au début de la requête, aucune arête n'a été mise à jour et aucun des nœuds n'est considéré comme atteignable (i.e. tous appartiennent à K_{na}). Ainsi, la première connexion se fait en reliant les deux nœuds de requête à cet ensemble (figure 2.6-1). Ensuite, un chemin valide est recherché et une portion du réseau est mise à jour (figure 2.6-2). Lors de l'étape suivante, les trois types de connexions L_{init}, L_{goal} et L_{na}, représentées respectivement sur les figures 2.6-3 a, b et c, sont envisagées. Dans l'exemple, la connexion L_{na} (figure 2.6-3c) permet de connecter l'ensemble K_{na} aux nœuds extrémaux et ainsi de trouver un chemin solution. Il est par ailleurs intéressant de noter que ce sont les arêtes bloquées qui étendent l'ensemble L_{na}, regroupant les nœuds non atteignables, alors qu'au contraire, la connexion des nœuds de requête tend à étendre les deux sous-ensembles K_{init} et K_{goal}.

Une fois qu'une connexion valide a été déterminée, un chemin solution peut être recherché au sein du réseau.

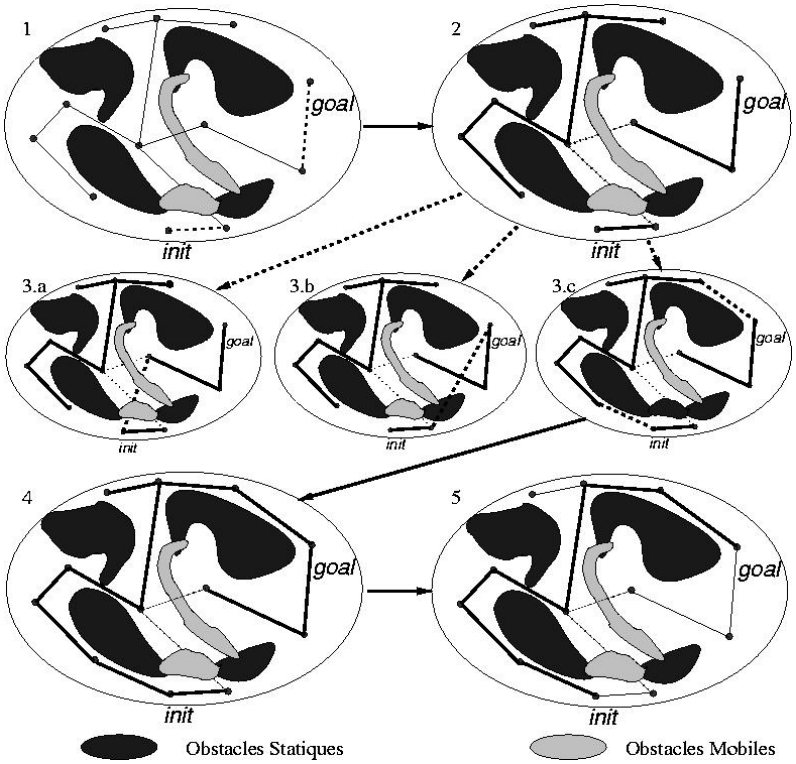


FIG. 2.6 – *Les nœuds de requête sont connectés au réseau (1) et une première tentative de recherche d'un chemin solution est réalisée (2). Si elle échoue, trois ensembles de nœuds apparaissent et trois types de connexions associées sont envisagés (3.a, 3.b, 3.c). Dans le cas considéré, le dernier type de connexion (3.c), permet de trouver un chemin valide (4,5).*

3.2 Recherche d'un chemin

Une fois que les nœuds de requête sont connectés au réseau, on cherche à extraire de ce réseau un chemin valide (bloc *Recherche d'un chemin* figure 2.3). L'algorithme *ExtractPath* associé à cette étape de recherche est représenté figure 2.7.

La fonction *BestPath* s'appuie sur l'algorithme A^* pour déterminer parmi les chemins non encore testés et reliant les nœuds de requête, celui correspondant à la distance la plus faible. Ensuite, sa validité vis à vis des obstacles mobiles est testée par la méthode d'étiquetage des arêtes décrite section 3.5. Si le chemin est valide, la requête est résolue sans calcul supplémentaire. Sinon le chemin est rajouté à la liste des chemins potentiels pour lesquels un processus de reconnexion sera testé (c.f. section 3.3). Ce processus de recherche est réalisé jusqu'à ce que tous les chemins potentiels aient été testés ou qu'une solution ait été trouvée.

Un exemple illustrant les différentes étapes de recherche/mise à jour est représenté figure 2.8. Après connexion des nœuds de requête (1), les différents chemins potentiels sont testés et

```
ExtractPath(G, Link)
 1   BrokenPath ← EmptyList
 2   Do
 3       Path ← BestPath(G, Link)
 4       PathLabeling ← Valid
 5       For All Edge in Path
 6           If EdgeLabeling(Edge)=Invalid
 7               PathLabeling ← Invalid
 8               AddToList(BrokenPath,Path)
 9           End If
10       End For All
11       If PathLabeling = Valid
12           Return Path
14       End If
13  While Path ≠ ∅
14  UpdateK(G)
15  Return BrokenPath
```

FIG. 2.7 – *Algorithme ExtractPath, recherchant un chemin valide au sein du réseau*

les arêtes associées encore non testées sont mises à jour (2 à 9). Comme aucun des chemins ne conduit à une solution, l'algorithme tente alors de reconnecter localement les portions de chemins invalides (10). Pour l'exemple présenté, c'est cette dernière étape qui permet au final d'obtenir une solution (11-12).

3.3 Reconnexions locales

Après chaque nouvelle connexion des nœuds de requête, l'algorithme tente de trouver un chemin solution ne comportant aucune arête bloquée. Lorsqu'un chemin valide n'existe pas, l'algorithme essaye de reconstruire localement un chemin valide en réparant certaines arêtes invalidées par les obstacles mobiles (bloc *Reconnexions locales* figure 2.3).

La stratégie privilégiée, visant à reconnecter les chemins invalidés seulement après avoir testé l'ensemble des chemins potentiels permet d'exploiter au maximum le réseau précalculé : alors que sa mise à jour ne nécessite des tests que vis à vis des obstacles mobiles, les opérations de reconnexion nécessitent également de prendre en compte les obstacles statiques. Les chemins comportant les portions invalides les plus courtes sont testés en premier, car ce sont les chemins le plus susceptibles d'être reconnectés.

La reconnexion des arêtes est effectuée par une méthode de diffusion de type RRT. Une version plus performante de cet algorithmme (RRT à Domaine Dynamique) fait l'objet du chapitre 3 qui donne une description détaillée de cette nouvelle technique de diffusion.

En dernier recours, si ce mécanisme de reconnexion échoue pour l'intégralité des chemins que l'on cherche à réparer, une nouvelle connexion des nœuds de requête est testée suivant la méthode décrite section 3.1.

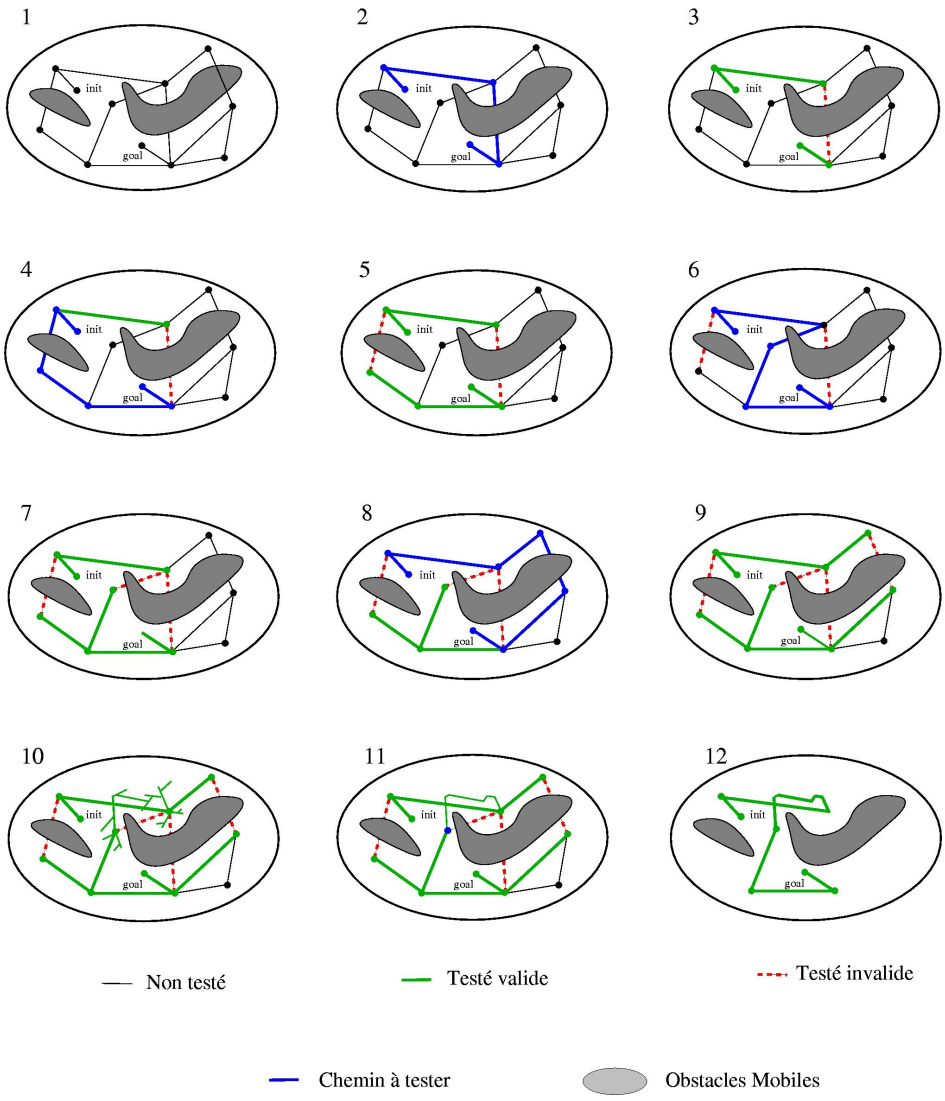


FIG. 2.8 – *Exemple de recherche d'un chemin solution. Une première connexion des nœuds de requête est effectuée (1) et tous les chemins potentiels sont testés (2 à 9). Comme aucun d'eux ne permet d'obtenir une solution, une méthode de reconnexion des chemins invalides est réalisée (10) permettant au final la résolution du problème (11-12).*

3.4 Renforcement du réseau

La méthode que nous avons présentée réalise deux opérations en parallèle : la mise à jour du réseau et la recherche d'un chemin solution. Si au final, le processus de recherche conduit à un étiquetage total du réseau sans aboutir à une solution, cela signifie que certaines arêtes invalidées par les obstacles mobiles rendent le réseau non connexe dans le contexte courant. Le réseau est alors renforcé en insérant des nœuds et des arêtes qui reconstruisent la connexité du réseau dans le contexte courant. Ce mécanisme de renforcement conduit à l'insertion de nouveaux cycles au niveau de la structure de données du réseau. Le principe utilisé pour reconstruire les composantes non connexes dans le contexte courant est le même que celui utilisé dans la visibilité [Siméon 00] : un nœud n'est ajouté que s'il permet de connecter deux composantes couramment invalides ou s'il n'est pas visible dans le contexte courant. Cette technique permet de limiter l'insertion des nœuds à ceux véritablement pertinents, ce qui conduit au renforcement du réseau par l'ajout de nouveaux cycles tout en conservant une structure de données de faible taille.

Notons également que cette phase de renforcement correspond à la phase la plus coûteuse du processus de résolution de requête. Pour cette raison, elle n'intervient seulement qu'en dernier lieu, quand toutes les opérations de mise à jour du réseau ont été mises en défaut.

3.5 Etiquetage des arêtes

Conjointement à la stratégie paresseuse de recherche/mise à jour, le planificateur utilise l'information des requêtes passées pour améliorer l'efficacité de l'étiquetage des arêtes. La mémorisation des tests est possible grâce à une structure de données spécifique associée à chaque arête (figure 2.9). Elle permet de sauvegarder les tests de validité de l'arête vis à vis des obstacles mobiles situés à des positions données.

L'étiquetage des arêtes est effectué au cours des tests successifs des chemins du réseau (ligne 6 de l'algorithme 2.7). Cette opération est détaillée à l'aide de l'algorithme figure 2.10. La première fois qu'une arête est étiquetée, aucune information n'a encore été stockée et les tests de validité s'effectuent normalement. Parallèlement, les résultats de ces tests sont stockés dans la structure de données (figure 2.9), en prenant comme positions clefs les positions courantes de chacun des obstacles mobiles. La prochaine fois que l'arête doit être mise à jour, avant de tester sa validité vis à vis de l'obstacle mobile, la position courante de cet obstacle est comparée aux positions déjà stockées. Si l'une d'elles correspond à la position actuelle, le résultat du test de collisions précédemment réalisé est extrait de la structure, évitant ainsi un nouveau test inutile. Sinon, la nouvelle position et le test de validité associé sont mémorisés. L'avantage de cette structure est que les tests sont mémorisés pour les obstacles mobiles considérés indépendamment. Ainsi, dès qu'un d'eux reprend un de ses anciens placements, les tests de mise à jour peuvent être allégés, même si le contexte dans sa globalité est nouveau.

En pratique, il convient cependant pour des raisons de place mémoire, de limiter le mécanisme de mémorisation aux dernières positions prises par les obstacles mobiles. Ce mécanisme s'avère très efficace pour éviter de nombreux tests inutiles pour les deux types de situations

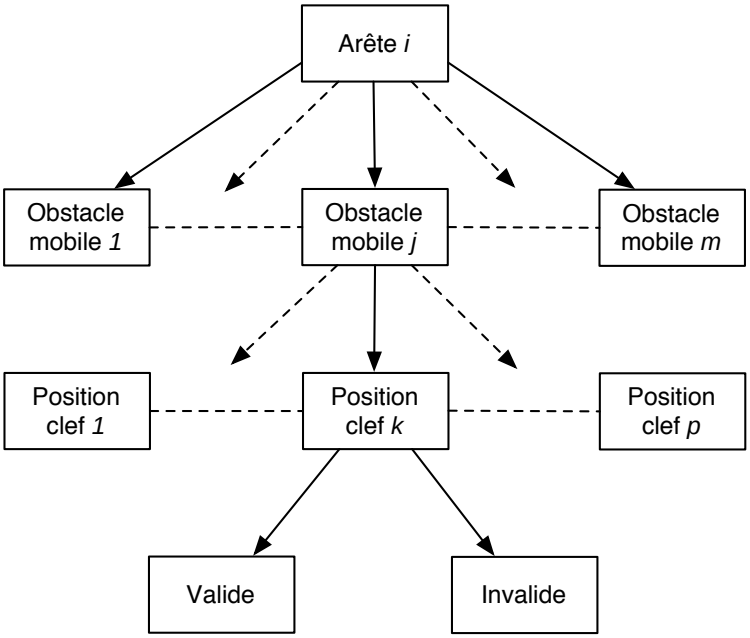


FIG. 2.9 – *Stockage des informations de collision au sein des arêtes*

suivants :

– Quand un obstacle possède un ensemble discret de positions (i.e. une porte soit fermée soit ouverte). Les tests de collisions ne sont réalisés qu'une seule fois par position discrète.

– Quand un obstacle s'arrête temporairement (i.e. un objet déplacé ou un robot à l'arrêt). La méthode de mémorisation permet alors de traiter l'obstacle, comme faisant partie de la scène statique.

Au final, pour une arête donnée les tests véritablement effectués sont ceux, difficilement évitables, associés à de nouvelles positions d'obstacles mobiles.

4 Résultats

Ce planificateur ainsi que les méthodes algorithmiques que nous présenterons chapitres 3 et 4, ont été implémentés en C sur la plate-forme *Move3D* [Siméon 01] développée au LAAS. Les temps de calcul correspondent à des simulations réalisées à partir d'une station de travail Sun Blade 100, 333 MHz sous SunOS 5.9. Dans ce chapitre, les valeurs indiquées dans les tableaux sont des moyennes sur 20 itérations. Les tests de collisions indiqués correspondent à chaque fois à la somme des tests statiques et dynamiques.

```
EDGE_LABELING(E)
 1   For each Movable Obstacle OM_i
 2       For each Key Position KP_j of OM_i stored in E
 3           If KP_j = CurrentPos
 4               IsCollision←ExtractColInfo(E,KP_j)
 5           Else
 6               IsCollision←TestCol(E, OM_i)
 7               Store IsCollision in R
 8           End If
 9           If IsCollision = True
10               Return invalid
11           End If
12       End For
13   End If
14   Return valid
```

FIG. 2.10 – *Algorithme d'étiquetage d'arête*

4.1 Performance globale

Dans ce premier test, le planificateur est comparé à deux autres méthodes :

– Un algorithme de diffusion *bi-RRT* [Kuffner 00]. Cette méthode de type simple requête n'utilise pas de réseau précalculé. Ainsi l'intégralité de l'information concernant l'espace libre est à reconstruire à chacune des requêtes.

– Un planificateur basé sur un réseau précalculé à l'aide de la méthode de *Visibilité-PRM* [Siméon 00], mais étiquetant systématiquement toutes les arêtes et connectant les nœuds de requête à toutes les composantes formées (méthode naïve mentionnée section 2.2).

L'environnement de test correspond à la scène de type bureau, représentée figure 2.1. A chaque requête, chacune des deux portes centrales prend aléatoirement la position ouverte ou fermée. Trois autres obstacles mobiles sont positionnés aléatoirement autour de la table. Les résultats des tests pour les trois types de méthodes sont représentés figure 2.11.

	Requête Simple	Planificateur Naïf	Planificateur Dynamique
Temps (s)	4.3	1.1	0.2
Collisions	5697	1722	356
Arêtes	45	78	78
Arêtes testées	45	78	18

FIG. 2.11 – *Performance du planificateur pour environnements changeants comparativement à un planificateur simple requête et à un planificateur naïf, avec mise à jour de l'intégralité du réseau.*

On note d'abord que la méthode naïve, basée sur une mise à jour globale du réseau est plus performante que la méthode de simple requête. Cette observation permet de mettre en évidence l'utilité d'un réseau initial : même si plus d'arêtes sont testées dans le cas du planificateur naïf (78 contre 45), celles-ci ne le sont que par rapport aux obstacles mobiles, ce qui diminue significativement le coût global des tests de collisions (1 722 contre 5 697). Observons également le gain de performance avec notre méthode, comparativement à une approche naïve. Le nombre de tests est pratiquement divisé par 5, malgré la faible taille des réseaux pour l'exemple considéré.

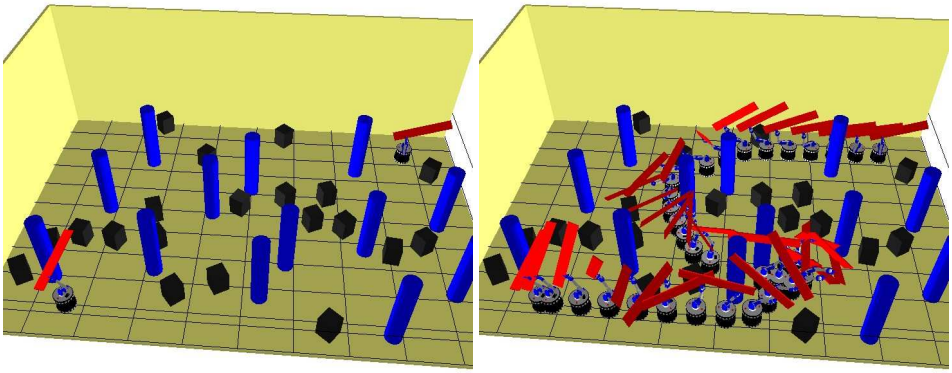


FIG. 2.12 – *Un exemple de requête (gauche) et un chemin solution (droite), pour une scène comportant des colonnes statiques et 20 obstacles mobiles à forme cubique.*

4.2 Mécanisme paresseux

Le but des tests suivants est d'évaluer l'influence de la méthode paresseuse de connexion/mise à jour sur les performances globales du planificateur. Pour cela, nous avons réalisé des tests pour un même environnement mais pour différentes tailles de réseaux initiaux (contenant éventuellement des cycles) et pour différentes quantités d'obstacles mobiles. L'environnement utilisé est représenté figure 2.12. Il se compose d'une plate-forme manipulatrice mobile évoluant dans une scène comportant 13 colonnes fixes et de 5 à 20 obstacles mobiles cubiques aléatoirement répartis.

Le nombre d'obstacles mobiles (OM) et les éléments caractéristiques des réseaux utilisés sont synthétisés ci-dessous :

Test	OM	Nœuds	Arêtes
A	5	100	99
B	5	100	318
C	20	100	99
D	20	100	318
E	20	200	700

Comme nous cherchons ici à évaluer la méthode paresseuse de connexion/mise à jour, les solutions impliquant des reconnexions locales ou un renforcement du réseau ne sont pas retenues.

Les performances sont encore une fois comparées à celles du planificateur naïf de la section 2.2. La figure 2.13 présente les résultats de ces tests.

Ces résultats mettent en évidence l'efficacité de la méthode, indépendamment des contextes considérés : pour tous les tests présentés, la méthode de connexion/mise à jour paresseuse, est de 7 à 12 fois plus rapide que celle basée sur un étiquetage global. Les situations les plus défavorables vis à vis de l'évaluation paresseuse apparaissent quand il est difficile d'obtenir un chemin solution, c'est à dire quand beaucoup d'obstacles mobiles sont présents et que le réseau est faiblement redondant (C,D). Même pour ces situations le gain de performance reste très conséquent.

	A	B	C	D	E
Méthode paresseuse					
Temps (s)	0.2	0.3	0.7	1.0	1.7
Collisions	340	478	640	681	1379
% d'arêtes testées	9.1	4.1	10.1	5.7	6.1
Planificateur naïf					
Temps (s)	2.4	3.6	4.8	7.3	15.7
Collisions	4602	7123	4882	6809	14338

FIG. 2.13 – *Influence de la méthode paresseuse de connexion/mise à jour en fonction du nombre d'obstacles mobiles et de la taille du réseau.*

4.3 Etiquetage des arêtes

L'environnement de la figure 2.12 est également utilisé pour tester l'influence du mécanisme de mémorisation sur le planificateur. Le réseau et les obstacles mobiles correspondent à ceux du test D de l'expérience précédente. La figure 2.14 indique le temps moyen d'étiquetage de l'ensemble du réseau en fonction du nombre d'obstacles mobiles déjà stockés dans la structure d'arête.

Comme on pouvait s'y attendre, le temps utilisé pour étiqueter les arêtes décroît linéairement en fonction du nombre de positions déjà stockées dans la structure de données. Dans le cas extrême où toutes les positions courantes des obstacles sont déjà mémorisées, aucun nouveau test de collisions n'est nécessaire pour la mise à jour des arêtes.

4.4 Renforcement du réseau

La figure 2.15 montre l'environnement utilisé pour illustrer le mécanisme de renforcement du réseau. Cet environnement de type maison possède une très grande complexité géométrique. Il est composé de nombreux objets qui nécessitent un total de plus de 70 000 facettes. En plus de cette scène statique, l'environnement comprend 7 portes mobiles. Le robot utilisé est représenté en haut de la figure 2.15. Afin de permettre le mécanisme de renforcement, le planificateur est

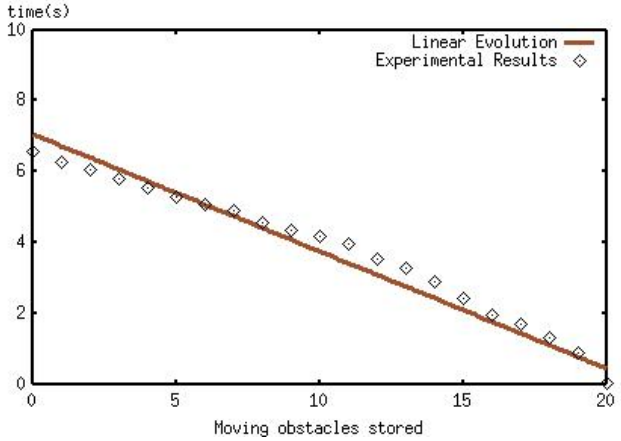


FIG. 2.14 – *Influence de la technique d´étiquetage d'arête : le temps de calcul décroît linéairement en fonction du nombre de positions déjà stockées.*

initialisé au moyen d'un arbre calculé par l'algorithme Visib-PRM [Siméon 00] (image en bas à gauche figure 2.15). Ensuite, 10 requêtes sont résolues à l'aide du planificateur, avec chacune des portes en position aléatoirement fermée ou ouverte. Quand la solution ne peut être trouvée à l'intérieur du réseau, même avec reconnexions locales, de nouveaux nœuds et arêtes sont rajoutés en utilisant de nouveau la technique Visib-PRM. L'image en bas à droite montre le réseau résultant après les différentes requêtes. Celui-ci a été renforcé à travers l'ajout de 3 nœuds et 11 arêtes qui participent à la création de plusieurs cycles nécessaires pour trouver un chemin quand certaines portes sont fermées. Sur l'ensemble des 10 requêtes, le temps total de renforcement est de 3.5 secondes. Une fois ce renforcement effectué, les différentes requêtes sont résolues pour un temps moyen de seulement 0.05 secondes (sans utiliser le mécanisme de mémorisation).

4.5 Déformation dynamique de chemins

Afin de mettre en évidence les capacités temps-réel du planificateur, nous avons également utilisé celui-ci pour contrôler des problèmes d'exécution de chemins. Ce mécanisme de contrôle a lieu de la façon suivante : le planificateur calcule d'abord un premier chemin valide dans le contexte courant. Ensuite, pour chaque intervalle de temps donné, le chemin est testé par rapport aux obstacles mobiles. Si celui-ci n'est plus valide, une tentative de reconnexion locale essaye de le réparer, comme cela avait été proposé pour les arêtes du réseau. Si elle échoue, notre planificateur est utilisé pour déterminer un nouveau chemin valide. Enfin, lorsque l'étape de mise à jour du chemin réussit, le robot avance d'un pas de longueur fixe. En pratique, seules les portions de chemin prochainement parcourues sont testées. Ceci permet d´éviter de replanifier dans le cas où un obstacle croise le chemin loin de la position courante du robot, alors que quand celui-ci arrivera sur la portion concernée, l'obstacle aura des chances de ne plus y être.

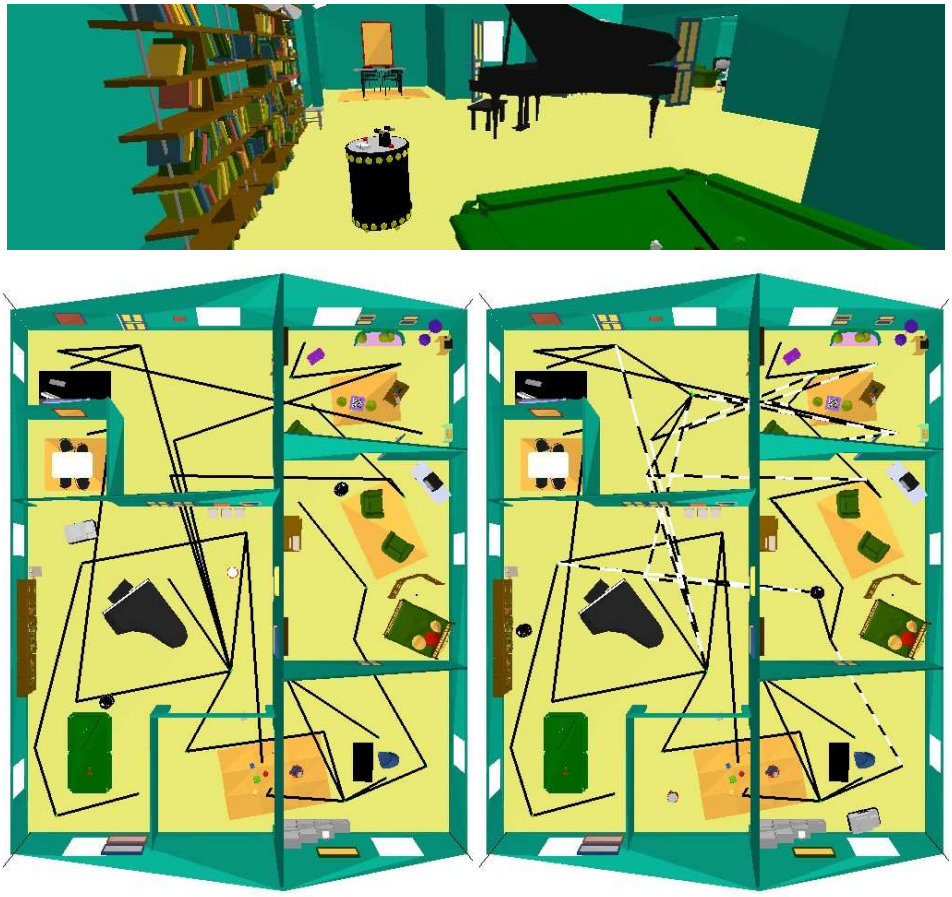


FIG. 2.15 – *Création de cycles au sein du réseau, pour un robot (en haut), évoluant dans un environnement de type maison. Un premier réseau ne comprenant aucun cycle est calculé (gauche). Après quelques requêtes, celui-ci est renforcé par l'apparition de nouveaux cycles (droite). Les arêtes en pointillé correspondent à des arêtes invalides dans le contexte courant.*

La figure 2.17 montre un exemple d'exécution de chemin contrôlé par le planificateur. Le robot mis en jeu est un bras manipulateur à 6 ddls, utilisé sur une ligne d'assemblage. Il est entouré de deux autres bras articulés jouant le rôle d'obstacles mobiles. La première image montre les positions initiales et finales du problème (la position finale est représentée en filaire), ainsi qu'un premier chemin calculé. Le robot évolue le long de ce chemin, mais il est perturbé successivement par le déplacement de chacun des deux autres bras présents dans la scène. Face à ces perturbations le planificateur est alors capable de reconstruire en temps-réel un chemin valide, donnant au robot la possibilité d'atteindre la position finale désirée.

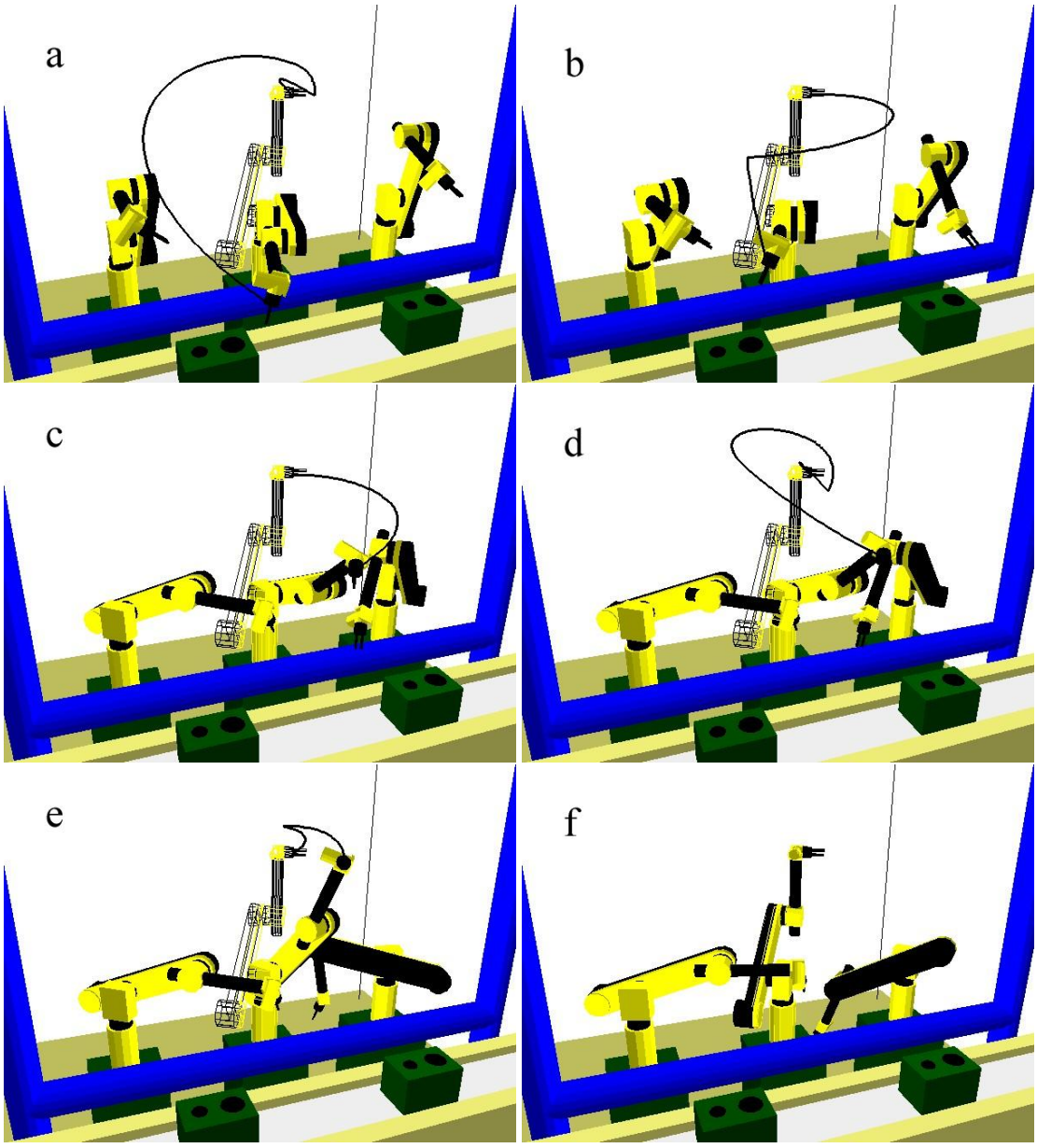


FIG. 2.16 – *Exécution de chemin contrôlée par le planificateur pour environnements changeants pour un bras manipulateur à 6 ddls utilisé sur une ligne d'assemblage.*

5 Choix du réseau initial

L'approche proposée s'appuie sur des réseaux de faible densité au sein de l'espace des configurations. Ce faisant, nous nous distinguons fortement des approches classiques de la littérature (c.f. [Leven 00, Bohlin 00, Nielsen 00]), qui utilisent des réseaux de grande densité et fortement redondants. Le choix d'un réseau peu dense se justifie d'une part par des temps de construction plus faibles et d'autre part par des opérations de mise à jour bien plus performantes qu'avec des réseaux de taille importante.

L'environnement de type maison à 7 portes présenté section 4.4 est utilisé de nouveau pour comparer les coûts de construction et mise à jour pour différents types de situations. Le tableau ci-dessous représente les temps de construction des réseaux, puis les temps de requêtes moyen, pour un robot circulaire à 2 ddls qui doit aller d'un coin de l'environnement à un autre, avec des positions de portes aléatoirement fermées ou ouvertes. Deux types de réseaux initiaux sont considérés : le premier G_{dense} (figure 2.17 a) est construit à l'aide d'une méthode standard de connexion aux k plus proches voisins et pour un couple ($N = 5000$, $k = 10$), avec N, nombre total de nœuds. Le deuxième réseau $G_{retract}$ (figure 2.17 b) est construit grâce à la méthode de rétraction présentée chapitre 4. Pour chaque réseau, deux types de mise à jour sont considérés : une mise à jour naïve avec étiquetage global des arêtes et une mise à jour paresseuse, à partir de la méthode décrite section 3.

	N	E	t init. (s)	t requête (s)	
				naif	paresseux
G_{dense}	5000	107 434	1633	17.	1.1
$G_{retract}$	113	170	164	0.2	0.05

On constate d'abord que la construction du réseau dense prend presque 10 fois plus de temps que la construction du réseau peu dense. De plus, pour les réseaux denses, les temps de mise à jour restent non négligeables même avec une méthode paresseuse ($t = 1.1$ secondes). Ce n'est qu'en combinant un méthode de mise à jour efficace avec l'utilisation de réseaux de faible densité, que l'on parvient à obtenir des temps de requêtes réellement temps-réel ($t = 0.05$ secondes).

En contrepartie de la plus grande efficacité de mise à jour, la connexité des réseaux que nous construisons est à priori moins robuste aux obstacles mobiles. Cette faiblesse est en réalité compensée par l'utilisation d'une méthode spécifique de reconnexion des portions invalides (c.f. chapitre 3). Grâce à cela, les chemins du réseau initial peuvent servir de routes principales, guidant la recherche de chemins solutions. De plus, la méthode des réseaux de rétraction (c.f. chapitre 4) et la méthode des réseaux robustes (c.f. chapitre 5) permettent de bénéficier de réseaux initiaux peu denses mais suffisamment riches pour pouvoir proposer différents chemins alternatifs en présence d'obstacles mobiles.

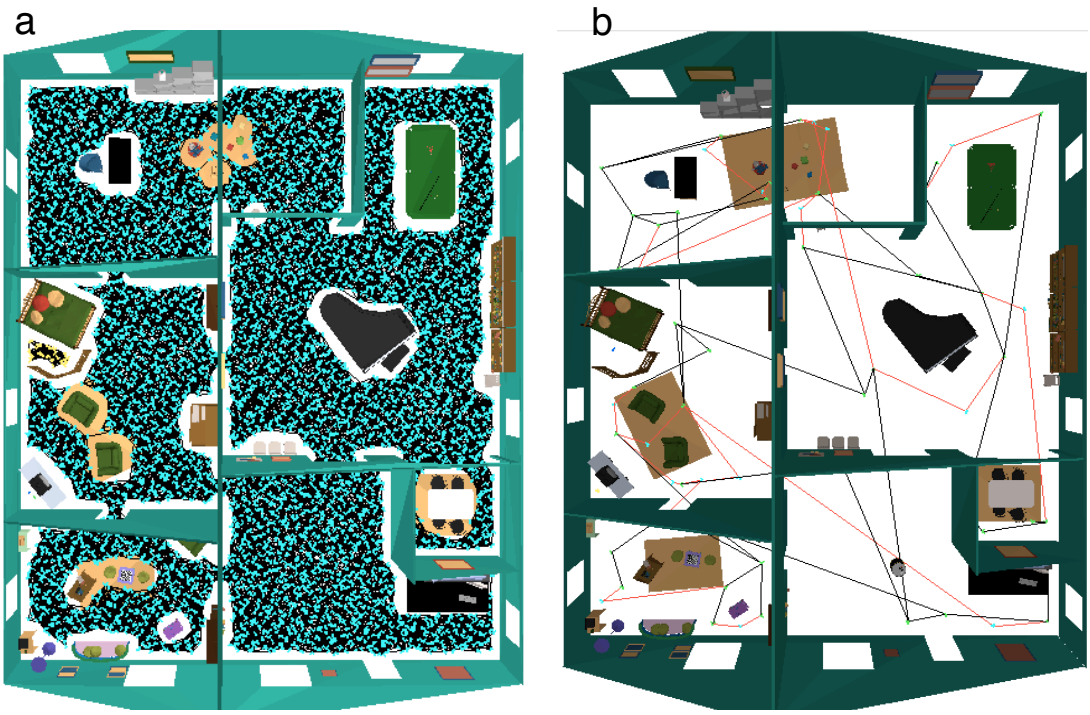


FIG. 2.17 – *Réseau dense (a) construit par une méthode PRM standard avec connexion aux k plus proches voisins et réseau peu dense (b) construit par la méthode de rétraction du chapitre 4. Un réseau de plus faible taille permet des opérations de mise à jour nettement plus performantes.*

Chapitre 3

Planificateur RRT à Domaine Dynamique

Au cours du chapitre 2, nous avons présenté un planificateur capable de résoudre avec efficacité des problèmes à environnements complexes, pour des contextes d'obstacles mobiles très variés. Un des éléments à l'origine des bonnes performances du planificateur est la faible taille des réseaux initiaux qui rend les opérations de mise à jour très rapides. En contrepartie, la connexité de ces réseaux est à priori moins robuste aux obstacles mobiles. Pour permettre de conserver l'efficacité de la méthode, nous proposons de réparer certaines portions du réseau quand cela est nécessaire. Parmi les techniques disponibles, on trouve les méthode de diffusion avec notamment la famille des algorithmes RRT (*Rapidly exploring Random Trees*). Bien que très performants pour un grand nombre de problèmes, l'algorithme RRT "de base" possède certaines faiblesses qui peuvent l'amener à explorer de façon peu efficace l'espace quand le domaine de tirage des configurations est mal adapté. Ce type de situation pourrait notamment apparaître si nous utilisions cette méthode comme outil de reconnexion locale.

Dans ce chapitre, après une analyse de l'algorithme RRT et des situations pour lesquelles son fonctionnement s'avère dégradé, nous décrirons une nouvelle méthode [Yershova 05, Jaillet 05] appelée *RRT à Domaines Dynamiques* ou encore *DD-RRT*, développée en collaboration avec l'équipe de S. Lavalle de l'UIUC. Les résultats décrits en fin de chapitre montrent la bien meilleure performance de cet algorithme pour traiter certaines situations pathologiques présentes dans les problèmes localement très contraints.

1 Analyse des RRT

1.1 Principe

Le principe général des RRT, introduits dans [LaValle 98] est le suivant. Partant d'une configuration initiale, l'algorithme explore incrémentalement l'espace des configurations jusqu'à trouver un chemin connectant la configuration initiale à la configuration finale. Le processus de construction de l'arbre est relativement simple. Une configuration q est tirée et le nœud le plus proche (pour une métrique donnée) est étendu vers cette configuration. Deux variantes d'extension existent pour cet algorithme. RRT-*Extend* tente, à chaque itération, d'étendre le nouveau nœud d'une distance fixée. Dans une variante plus "agressive", RRT-*Connect* [Kuffner 00], le pas d'extension est répété jusqu'à ce qu'il y ait collision ou que la configuration tirée soit atteinte. L'algorithme associé à cette version est représenté figure 3.1. Une version bidirectionnelle des RRT existe également. Elle consiste à construire simultanément deux arbres issus des deux configurations à relier et en appliquant alternativement l'étape de construction à chacun des arbres. Si l'on rajoute une contrainte de façon à maintenir la taille équilibrée (en nombre de nœuds), on parle alors d'algorithme *RRT bidirectionnel équilibré*.

```
BUILD_RRT(q_init)
 1  T.init(q_init);
 2  for k = 1 to K do
 3      q_rand ← RANDOM_CONFIG();
 4      q_near ← NEAREST_NEIGHBOR(q_rand, T);
 5      if CONNECT(T, q_rand, q_near, q_new)
 6          T.add_vertex(q_new);
 7          T.add_edge(q_near, q_new);
 8  Return T;
```

FIG. 3.1 – *Algorithme de construction de RRT-CONNECT.*

1.2 Exploration et biais de Voronoï

Les capacités d'exploration des algorithmes RRT s'expliquent en considérant le diagramme de Voronoï des nœuds de l'arbre (c.f. figure 3.2). Comme la sélection des nœuds se fait sur le critère du plus proche voisin, la probabilité pour un nœud d'être choisi pour l'extension est proportionnelle au volume occupé par sa région de Voronoï. Ainsi, le processus normal de construction d'un arbre RRT se fait en ajoutant des nœuds qui divisent les régions de Voronoï les plus vastes en régions plus petites. Dans la majorité des cas, cette propriété permet aux arbres RRT de s'étendre rapidement vers les régions encore inexplorées de l'espace des configurations.

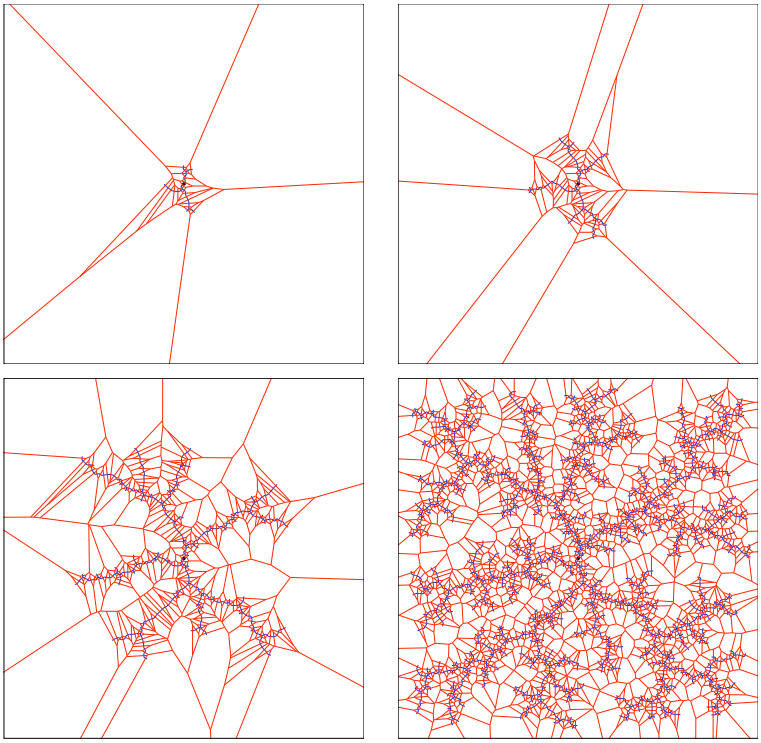


FIG. 3.2 – *Le processus d'expansion est guidé par le biais de Voronoï : les nœuds associés aux régions de Voronoï les plus grandes ont une plus forte probabilité d'être sélectionnés pour l'extension (d'après [LaValle 01b]).*

1.3 Exploration versus affinage

Considérons un arbre construit à l'aide des RRT et le diagramme de Voronoï qui lui est associé (figure 3.3). Les régions de Voronoï dont la taille est limitée par les bornes de l'espace des configurations, sont associées à des nœuds que l'on nomme *nœuds frontières*. Ils permettent de définir deux zones complémentaires de l'espace des configurations : une *zone intérieure* (figure 3.3-a) et une *zone extérieure* (figure 3.3-b) distinguant les régions de Voronoï des nœuds non frontières de celles associées aux nœuds frontières.

On peut alors décomposer le processus d'expansion des RRT en deux principaux modes qui dépendent des zones dans lesquelles se situent les différentes configurations échantillonnées :

– Un *mode d'affinage* (figure 3.3-a) associé aux échantillons situés dans la zone intérieure. L'algorithme contribue alors, par l'ajout des nœuds, à affiner le modèle de $\mathcal{C}_{free}$ au sein des régions déjà couvertes par l'arbre. En ce sens ce mode de recherche se rapproche de l'idée des recherches multirésolution : la résolution courante (i.e. la densité de nœuds), est augmentée jusqu'à obtenir une précision du modèle de l'espace libre suffisante pour obtenir

la solution.

– Un *mode exploratoire* (figure 3.3-b), associé aux échantillons appartenant à la zone extérieure. Contrairement au précédent, ce mode tend à guider l'extension de l'arbre en direction des portions encore inexplorées de $\mathcal{C}$.

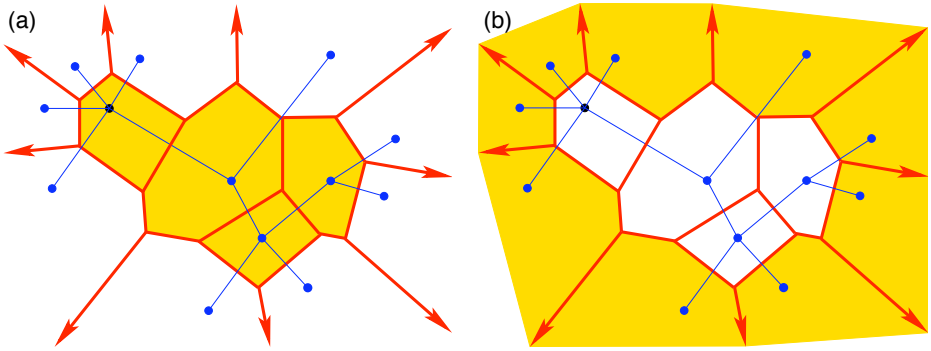


FIG. 3.3 – *Deux modes d'expansion pour deux régions caractéristiques de l'espace : l'affinage de régions déjà explorées quand l'échantillon appartient à la zone intérieure (a) et l'exploration de nouvelles régions quand il se trouve dans la zone extérieure (b).*

1.4 Exemple de situation pathologique

La figure 3.4 montre un exemple de situation qui illustre certains problèmes induits par le guidage Voronoï. Dans chacune des 4 situations de la figure, le but est d'extraire un robot ponctuel hors d'une forme géométrique appelée *bug trap*[1]. La faible taille de l'orifice de sortie, relativement à l'espace libre à l'intérieur de la forme, caractérise un problème dit "à passage étroit", qui reste un problème difficile, même pour les planificateurs probabilistes.

Sur la figure 3.4-a, la taille de la région intérieure associée aux nœuds non frontières, est comparable à celle de la région extérieure. Ceci laisse de bonnes chances d'échantillonner suffisamment à l'intérieur du bug trap pour réussir finalement à trouver un chemin solution. La complexité du problème est largement amplifiée, lorsque le domaine de tirage des configurations est agrandi (figure 3.4-b). Les nœuds situés en bordure du piège vont avoir cette fois-ci une probabilité d'être sélectionnés beaucoup plus grande que les nœuds situés à l'intérieur. La difficulté peut être telle, que l'algorithme RRT ne soit pas capable de résoudre le problème en des temps raisonnables (voir les résultats figure 3.12 section 4). Dans cet exemple, la majeure partie des tentatives réalisées pour développer l'arbre va correspondre au mode exploratoire : de nombreux essais d'expansion des nœuds frontières de l'arbre sont effectués alors qu'il est

[1]Le nom *bug trap* qui en anglais signifie *piège à insectes*, est utilisé car la forme de l'obstacle est similaire à celle de pièges utilisés pour capturer des insectes volants.

nécessaire d'affiner le modèle des portions déjà explorées de l'espace libre, avant de pouvoir poursuivre l'exploration.

1.5 Nœuds frontières et nœuds bordants

Il est important de noter que ce sont les nœuds frontières qui sont à l'origine du biais important vers l'espace encore inexploré. Dans beaucoup de cas, ce biais est intéressant, car il permet à l'arbre de s'étendre rapidement. Cependant il peut aussi, comme dans l'exemple précédent, conduire à une perte de performance. Cette situation se produit lorsqu'un nœud frontière est également un nœud situé à proximité d'un obstacle. Ce type de nœuds est dit *bordant*. Dans l'exemple de la figure 3.4, on constate que tous les nœuds frontières sont aussi des nœuds bordants. Le problème est que ces nœuds ont une forte probabilité d'être sélectionnés pour être étendus en direction des obstacles, alors que la plupart du temps leur extension ne pourra se faire.

En général, l'algorithme attribue donc aux nœuds bordants un biais de Voronoï disproportionné par rapport à l'espace qu'ils peuvent effectivement explorer. Ceci se traduit par des tentatives infructueuses d'ajout de nœuds, chaque tentative nécessitant des calculs coûteux : recherche du nœud le plus proche, calcul du chemin interpolant les deux configurations et tests de collisions le long du chemin. Le coût de ces tests augmente donc avec la complexité géométrique de la scène. De plus, pour des systèmes soumis à des contraintes de fermeture cinématique [Cortés 04], des calculs de cinématique inverse viennent se rajouter. Enfin pour des systèmes non-holonomes, des opérations d'intégration numérique peuvent également être nécessaires pour le calcul des chemins locaux. Il est donc important de limiter ces tentatives de développement infructueuses qui peuvent dégrader fortement le développement de l'arbre.

1.6 Sensibilité aux bornes de $\mathcal{C}$

Un autre point important, illustré par l'exemple de la figure 3.4, est la dépendance de l'algorithme vis à vis des bornes de l'espace des configurations. En effet, la répartition des zones contraintes suivant les différentes directions de l'espace peut être très inégale. Si pour un problème donné, les bornes de l'espace sont mal adaptées par rapport aux directions contraintes, les performances seront aussi fortement affectées. Dans la figure 3.4(a), les limites de l'environnement sont suffisamment proches de l'espace couvert par l'arbre pour permettre au planificateur d'affiner l'arbre existant, et conduire ainsi à la résolution du problème avec une bonne performance. Au contraire, le domaine d'échantillonnage de l'exemple (b) est largement supérieur à l'espace couvert par l'arbre, ce qui va entraîner de nombreuses tentatives d'expansion des nœuds bordants en direction des obstacles. Les exemples (c) et (d) illustrent d'autres types de situations pour lesquelles cette fois-ci l'exploration est arbitrairement biaisée selon une direction donnée de l'espace. En pratique, ce type de situation peut fréquemment arriver. En effet, la forme des régions inexplorées de $\mathcal{C}_{free}$ peut être très différente de celle de $\mathcal{C}$, attribuant un biais mal réparti selon chacune des directions de l'espace. Ceci est d'autant plus notable dans des espaces de dimension élevée. En particulier, ce phénomène de dépendance in-

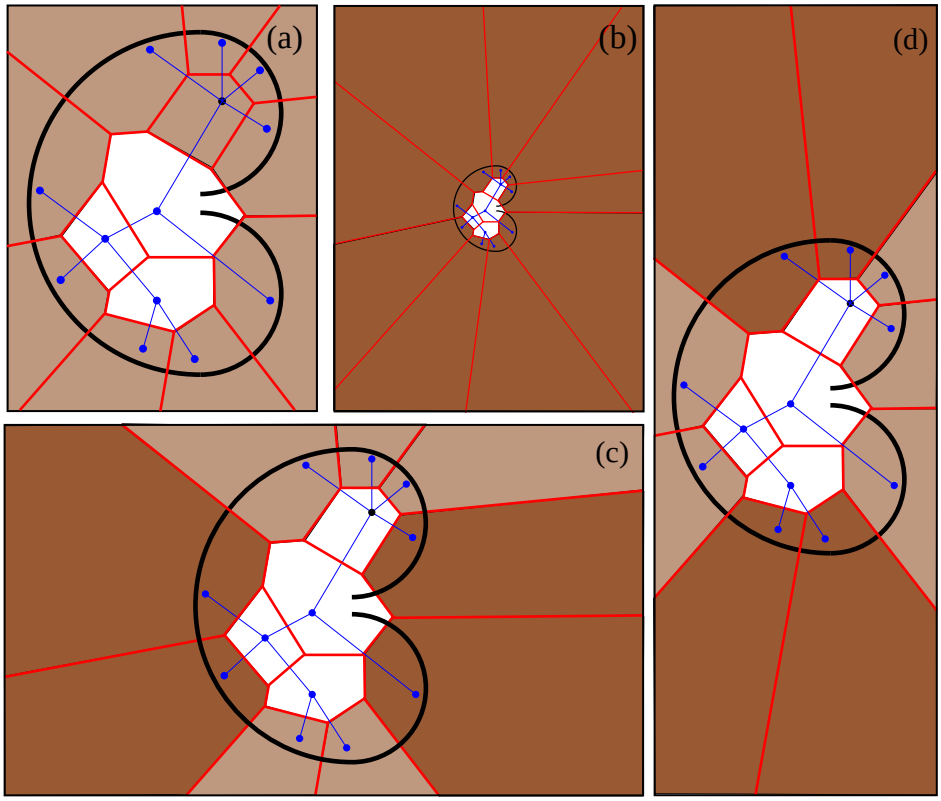


FIG. 3.4 – *Pour l'algorithme RRT basique, la région de Voronoï des nœuds frontières est dépendante de la taille de $\mathcal{C}$. Ainsi, en fonction des bornes choisies, le biais attribué à l'exploration de régions encore inexplorées peut être faible (a), fort (b), ou encore orienté arbitrairement selon certaines directions (c et d).*

tervient fortement dans les problèmes pour lesquels rotations et translations nécessitent d'être combinées pour trouver un chemin solution. Dans ce cas, le biais attribué à ces différents types de degrés de liberté peut se faire de manière très inégale. Ces situations sont cependant très difficiles à analyser, car elles font de plus intervenir la métrique choisie qui attribue déjà une pondération différente entre rotations et translations.

La motivation de cet algorithme est donc double : il s'agit d'une part de trouver une méthode qui réduise la dépendance vis à vis des bornes de l'espace des configurations. D'autre part, il s'agit de proposer une méthode plus performante, en réduisant le nombre de tentatives d'expansion infructueuses en présence de nombreux nœuds frontières bordants. La force de notre algorithme est de pouvoir contrôler les deux principaux modes d'expansion en jouant notamment sur le biais de Voronoï associé aux nœuds. La section suivante définit de façon formelle

le problème.

1.7 Formulation du problème

Considérons N un ensemble de nœuds de l'espace libre C_{free} et D le diagramme de Voronoï de N.

Définition 1.1 *Soit $\mathscr{L}$, une méthode locale permettant de calculer un chemin $\mathscr{L}(n,n')$ entre deux nœuds n et n'. Nous définissons alors le* ***domaine de visibilité*** *d'un nœud n pour un $\mathscr{L}$ donné, de façon similaire à [Siméon 00] :*

$$Vis_{\mathscr{L}}(n) = \{n' \in C_{free} \text{ tels que } \mathscr{L}(n,n') \in C_{free}\}$$

Définition 1.2 *Pour un nœud $n \in N$ et sa région de Voronoï $D(n)$, nous définissons la* ***région de Voronoï visible*** *de n comme étant $O(n) = Vis_{\mathscr{L}}(n) \cap D(n)$.*

Considérons enfin $\bigcup_{n \in N} O(n)$, union de toutes ces régions de Voronoï visibles que l'on appelle *domaine de Voronoï visible*. Ces régions vérifient la propriété suivante : toute configuration à l'intérieur de ce domaine peut être connectée (i.e. il n'y a pas de collision) au nœud le plus proche de l'arbre (puisqu'elle appartient à son domaine de visibilité). Un échantillonnage dans ces régions permettrait donc de diminuer fortement le biais en direction des obstacles tout en conservant un fort pouvoir d'expansion en direction de l'espace inexploré. Pour cette raison, un échantillonnage à l'intérieur de ce domaine de Voronoï visible serait tout à fait adapté. Il permettrait de combiner l'avantage d'un échantillonnage guidé par le biais de Voronoï, tout en limitant les situations de blocage grâce à la prise en compte des obstacles dans le calcul des régions de Voronoï.

Par ailleurs, il serait intéressant que la répartition des points dans le domaine de Voronoï visible soit non uniforme. Ceci permettrait de concentrer les tirages dans les zones à passages étroits, plutôt que dans les régions non contraintes de l'espace. Le calcul d'une telle distribution n'est cependant pas imaginable en raison de sa complexité prohibitive. Dans la section suivante, nous proposerons donc d'utiliser un échantillonnage basé sur une approximation de ces domaines de Voronoï visibles. Cette approximation présente l'avantage d'être très simple à calculer tout en conservant l'intérêt des propriétés décrites ci-dessus.

2 Algorithme DD-RRT

2.1 Domaine Dynamique

Afin de contourner la complexité prohibitive du calcul exact des domaines de Voronoï visibles, nous introduisons une version approchée de ces domaines, définie de la façon suivante :

Définition 2.1 *Étant donné un nœud bordant n situé à une distance d'au plus ε de C_{obs}, on définit le* ***domaine de bordure*** *de n par l'intersection entre sa région de Voronoï avec une boule de l'espace des configurations centrée sur n et de rayon R.*

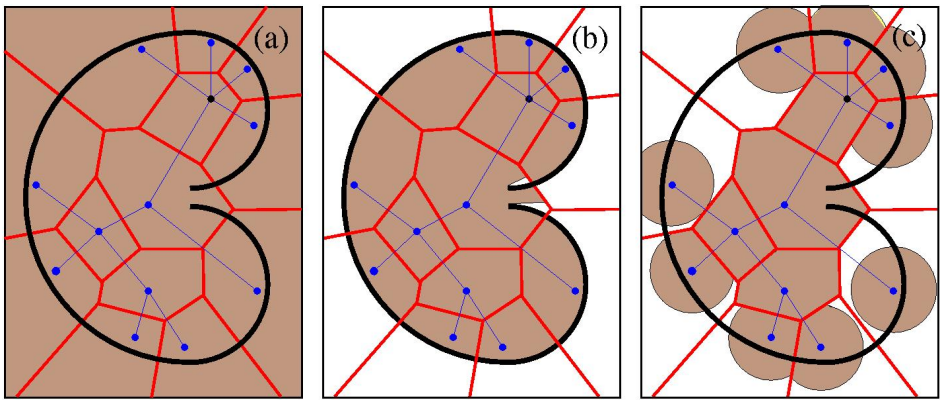


FIG. 3.5 – *Différents domaines d'échantillonnage sont présentés pour un ensemble de nœuds situés à l'intérieur d'un bug trap : (a) domaine d'échantillonnage classique des RRT, (b) région de Voronoï visible, (c) domaine dynamique.*

Définition 2.2 *Le **domaine dynamique** de rayon R pour un ensemble de nœuds N, correspond à l'union des domaines de bordure des nœuds bordants, combinée à l'union des régions de Voronoï des nœuds non bordants. Une distribution uniforme sur ce domaine est appelée **distribution à domaine dynamique**.*

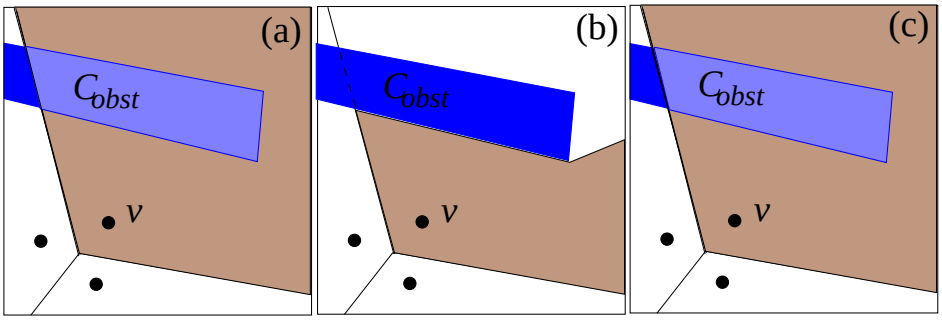


FIG. 3.6 – *Domaines d'attraction possibles pour un nœud non bordant (i.e. loin des obstacles) : (a) région de Voronoï pour les RRT standards (b) région de Voronoï visible, (c) domaine dynamique.*

La figure 3.5 illustre sur l'exemple bug trap la différence entre le domaine d'échantillonnage des RRT, le domaine de visibilité de Voronoï et le domaine dynamique pour un arbre RRT. Lorsqu'un nœud est suffisamment loin d'un obstacle (i.e. non bordant), son domaine d'attraction se confond avec la région de Voronoï des RRT classiques (figure 3.6). Par contre, lorsqu'un nœud est bordant, son domaine d'attraction se limite au domaine de bordure permettant ainsi

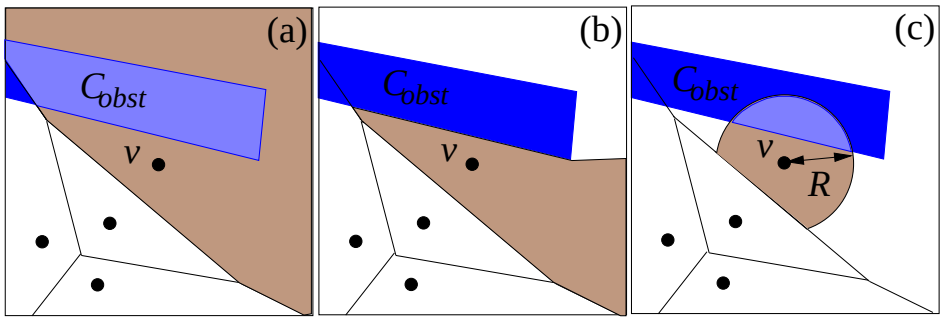


FIG. 3.7 – *Domaines d'attraction possibles pour un nœud bordant (i.e. près des obstacles) : (a) région de Voronoï pour les RRT standards, (b) région de Voronoï visible, (c) domaine dynamique.*

de compenser la présence de l'obstacle (figure 3.7).

```
BUILD_DYNAMIC_DOMAIN_RRT(q_init)
 1   T.init(q_init);
 2   for k = 1 to K do
 3       repeat
 4           q_rand ← RANDOM_CONFIG();
 5           q_near ← NEAR_NEIGH(q_rand, T, d_near);
 6       until (d_near < q_near.radius)
 7       if q_new ← CONNECT(T, q_rand, q_near)
 8           q_new.radius = ∞;
 9           T.add_vertex(q_new);
10           T.add_edge(q_near, q_new);
11       else
12           q_near.radius = R;
13   Return T;
```

FIG. 3.8 – *Algorithme RRT à Domaine Dynamique.*

2.2 Algorithme

Afin d'obtenir un échantillonage dans le domaine dynamique, l'idée principale de l'algorithme DD-RRT est de calculer dans un premier temps une distribution uniforme dans un hypercube contenant ce domaine, puis de restreindre cette distribution au domaine dynamique. En pratique, l'hypercube est calculé à partir de la boîte englobante de l'ensemble des domaines de bordure. Partant d'une distribution initiale uniforme, la distribution obtenue dans le domaine

restreint est également uniforme.

Le calcul de cette restriction correspond aux lignes 3 à 6 de l'algorithme DD-RRT présenté figure 3.8. Une variable *radius* est associée à chacun des nœuds de l'arbre. Cette variable prend R comme valeur dans le cas où le nœud est un nœud bordant et ∞ dans le cas contraire. Dans un premier temps, la valeur R est considérée constante, calculée lors de l'initialisation de l'algorithme. Dans la section 3, nous proposerons une amélioration de cet algorithme, permettant d'adapter au cours de la recherche la valeur R attribuée à chaque nœud.

Le pseudo-code de la figure 3.8 montre l'intégration de l'échantillonnage par domaine dynamique au sein de l'algorithme RRT-connect. La mise à jour du rayon associé aux nœuds bordants se fait à la volée. Au début de l'exploration, tous les nœuds sont considérés comme non bordants. Dès qu'une tentative d'expansion d'un nœud échoue (ce qui signifie que sa distance aux obstacles est inférieure à ε), le point devient un nœud bordant . La variable *radius* de ce nœud est alors affectée de la valeur R (lignes 11-12 du pseudo-code). Ainsi, la zone d'attraction du nœud sera restreinte à son domaine de bordure pour les échantillonnages suivants.

Les RRT à domaines dynamiques conservent la propriété de complétude probabiliste propre aux planificateurs aléatoires. La justification de cette propriété découle directement de celle présentée dans [Kuffner 00] pour les RRT.

Une autre remarque concerne l'influence de la valeur de la variable *radius*, sur le comportement de l'algorithme. A l'initialisation ($radius = \infty$), l'algorithme DD-RRT se comporte de la même manière que l'algorithme RRT, en privilégiant le mode exploratoire. Par la suite, les domaines dynamiques se réduisant dans les régions proches des obstacles, l'algorithme favorise le développement de nœuds dans les régions contraintes de l'environnement. Le mode d'affinage est donc renforcé étant donné que le biais des nœuds au contact est réduit.

De façon générale, l'algorithme DD-RRT tend à augmenter le nombre de configurations échantillonnées et à solliciter davantage la recherche de plus proches voisins, au profit d'une diminution des tentatives d'extension infructueuses dans le développement de l'arbre. Or ce sont ces dernières opérations qui sont les plus coûteuses (notamment à cause des tests de collisions qu'elles nécessitent). Afin d'augmenter les performances de cet algorithme, il est cependant intéressant d'utiliser les méthodes efficaces proposées dans [Atramentov 02] pour les calculs des plus proches voisins, qui sont aussi des opérations coûteuses lorsque l'arbre atteint une taille importante.

2.3 Analyse des performances

Dans cette section, nous proposons une analyse comparative de la performance de DD-RRT, avec celle de l'algorithme RRT standard. Considérons les notations suivantes :

- t^{near} : temps de calcul du nœud le plus proche.
- t^{exp} : temps du test de validité d'un chemin.
- t^{shoot} : temps du calcul d'un échantillon dans $\mathcal{C}$.
- p_{DD}^{near} : probabilité qu'un échantillon tiré dans la boîte englobante appartienne au domaine dynamique.
- p_{DD}^{exp} : probabilité d'extension d'un échantillon calculé pour le domaine dynamique.

– p_{RRT}^{exp} : probabilité d'extension réussie d'un nœud de $\mathcal{C}$.
– T_{DD} : temps moyen d'insertion d'un nœud en utilisant l'algorithme DD-RRT.
– T_{RRT} : temps moyen d'insertion d'un nœud en utilisant l'algorithme RRT.

On peut alors établir pour les deux méthodes considérées, les temps moyens d'insertion d'un nouveau nœud (en négligeant le coût des opérations de mise à jour des structures de données) :

$$T_{DD} = t^{shoot} + \frac{t^{near}}{p_{DD}^{near}} + \frac{t^{exp}}{p_{DD}^{exp}} \quad \text{et} \quad T_{RRT} = t^{shoot} + t^{near} + \frac{t^{exp}}{p_{RRT}^{exp}}$$

A nombre de nœuds égal, pour que la méthode des domaines dynamiques soit plus efficace que l'algorithme RRT , il faut que $T_{DD} < T_{RRT}$ c'est à dire :

$$\frac{t^{near}}{p_{DD}^{near}} + \frac{t^{exp}}{p_{DD}^{exp}} < t^{near} + \frac{t^{exp}}{p_{RRT}^{exp}}$$

qui peut se reformuler de la façon suivante :

$$\left(\frac{t^{near}}{t^{exp}}\right) \cdot \left(\frac{1 - p_{DD}^{near}}{p_{DD}^{near}}\right) \cdot \left(\frac{p_{DD}^{exp} \cdot p_{RRT}^{exp}}{p_{DD}^{exp} - p_{RRT}^{exp}}\right) < 1$$

L'équation ci-dessus permet d'expliciter dans quelles conditions la méthode des domaines dynamiques sera plus avantageuse que la méthode standard, à savoir :

– Quand le coût des tests de collisions est largement prédominant comparativement au coût associé à la recherche du nœud le plus proche ($\frac{t^{near}}{t^{exp}} \ll 1$). C'est en particulier le cas lorsque l'environnement a une forte complexité géométrique.
– Quand la boîte englobante utilisée pour limiter l'échantillonnage approxime correctement le domaine dynamique ($p_{DD}^{near} \simeq 1$).
– Quand le problème est fortement contraint et donc que les nœuds sont difficiles à étendre ($p_{DD}^{exp} \cdot p_{RRT}^{exp} \ll 1$) et quand l'utilisation du domaine dynamique augmente fortement les chances d'extension ($p_{RRT}^{exp} \ll p_{DD}^{exp}$).

C'est donc pour les problèmes contraints, à forte complexité géométrique correspondant aux problèmes difficiles pour les méthodes probabilistes, que l'algorithme DD-RRT apporte un gain de performance le plus notable.

3 Algorithme DD-RRT adaptatif

3.1 Influence du rayon des domaines dynamiques

Comme nous l'avons vu précédemment, le développement des arbres RRT se fait selon deux modes d'expansion induits par le biais de Voronoï. L'intérêt de l'algorithme DD-RRT est de permettre le contrôle efficace du biais d'exploration relatif à ces deux modes. Le contrôle entre ces modes se fait grâce à la valeur R du paramètre *radius* : plus le rayon est grand et plus l'algorithme tend à privilégier l'exploration. Au contraire, plus il est petit et plus l'affinage des régions déjà explorées prédomine.

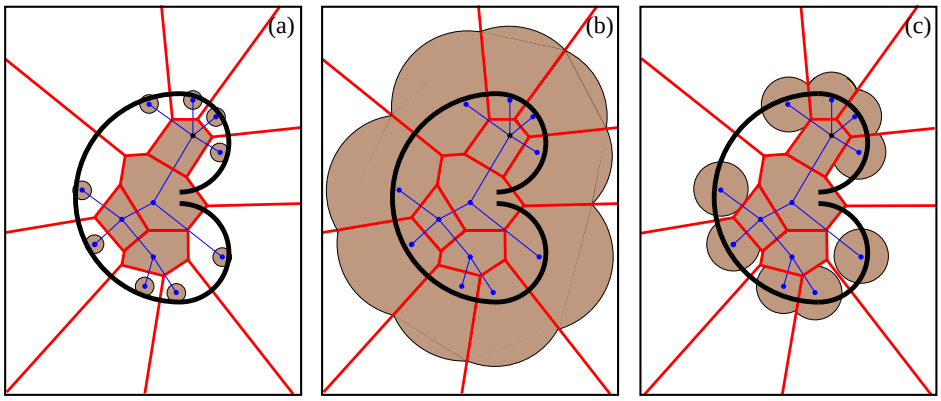


FIG. 3.9 – *Domaines dynamiques associés à 3 valeurs R différentes. (a) un faible rayon biaise l'expansion vers un affinage des régions déjà explorées, (b) avec un fort rayon le mode exploratoire prédomine, (c) exploration et affinage sont davantage équilibrés.*

Pour un problème donné, l'équilibre entre ces deux types de comportements peut cependant être délicat à déterminer. Un poids trop important attribué à l'affinage peut conduire à un nombre de nœuds trop élevé, alors que l'exploration de nouvelles régions est préférable. A l'opposé, un comportement dominé par l'exploration peut conduire à de nombreux essais infructueux d'extension de l'arbre vers les régions extérieures. En effet, l'affinage de régions déjà explorées peut s'avérer nécessaire pour découvrir de nouveaux passages et ainsi permettre la poursuite de l'extension de l'arbre. Le contrôle de la valeur R est donc un élément important pour la performance du planificateur.

Comme illustré par la figure 3.9, l'équilibre entre les modes d'expansion est obtenu lorsque les domaines calculés approchent au mieux les domaines de Voronoï visibles. Dans la version de l'algorithme DD-RRT décrite section précédente, le rayon attribué à chaque nœud ne peut prendre que deux valeurs (R ou ∞), alors que le domaine de Voronoï visible d'un nœud bordant peut beaucoup varier suivant que la région explorée est localement peu ou fortement contrainte par les obstacles de $\mathcal{C}$.

L'amélioration décrite dans cette section vise à adapter le rayon de chacun des nœuds au cours de la recherche, en fonction de l'espace libre. Elle permet un équilibrage automatique des modes d'affinage et d'exploration pendant la recherche.

3.2 Méthode de réglage adaptatif

La figure 3.10 présente l'intérêt d'un réglage différent de la valeur de *radius* dans deux situations. Dans le premier cas (en haut), bien que le nœud soit à proximité d'un obstacle, sa région de Voronoï visible reste large (a) et serait mieux approchée par un domaine de bordure de rayon important (b) évitant un affinage excessif de cette région. Dans le second cas (en bas), le nœud est fortement contraint pas les obstacles et en conséquence la région de Voronoï associée

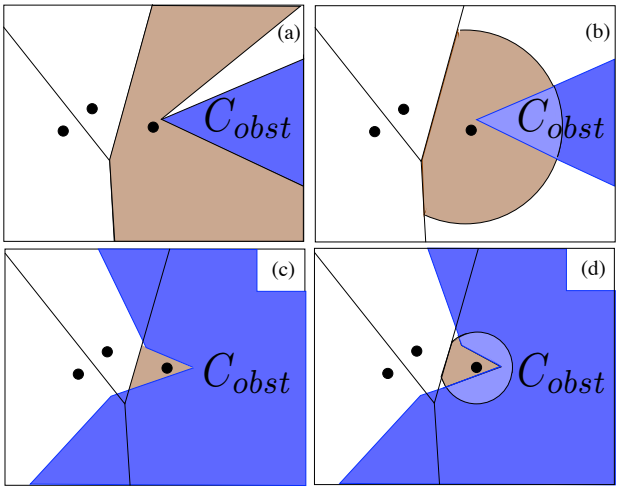


FIG. 3.10 – *Régions de Voronoï visibles (a-c) et domaines de bordure (b-d) associés à un nœud bordant dans le cas d'une région faiblement contrainte par les obstacles (fig. du haut), ou fortement contrainte (fig. du bas).*

est petite (c). Dans ce cas, un domaine de bordure associé à un petit rayon (d) est préférable afin de privilégier l'affinage dans les passages étroits.

Le problème est maintenant de déterminer dans quelle situation se trouve un nœud sans pour autant connaître la forme explicite des obstacles de $\mathcal{C}$. La solution que nous proposons est d'utiliser l'information obtenue lors des différentes tentatives d'expansion d'un nœud pour obtenir une meilleure évaluation de son domaine de visibilité de Voronoï. Chaque échec d'extension d'un nœud augmente sa probabilité d'être fortement entouré d'obstacles. Au contraire, à chaque fois qu'une extension réussit, cette probabilité décroît. Par conséquent, nous proposons d'adapter la taille de son domaine dynamique en fonction du résultat de ses tentatives d'expansion.

Le pseudo-code représentant l'algorithme DD-RRT adaptatif est représenté figure 3.11. A partir du moment où l'expansion d'un nœud échoue, il devient bordant et son rayon est initialisé à une valeur donnée (le choix de cette valeur sera discuté en fin de section 4.2). Ensuite, à chaque fois que ce nœud est sélectionné pour une tentative d'expansion, son rayon est modifié : lorsque l'expansion réussit (conduisant à la création d'un nouveau nœud de l'arbre), la valeur du rayon augmente d'un pourcentage α (ligne 10 de l'algorithme). Lorsqu'elle échoue, la valeur du rayon diminue du même ratio (ligne 17 de l'algorithme).

Afin de conserver la complétude probabiliste de l'algorithme, il convient de maintenir la possibilité pour un nœud d'être étendu. Pour cela, on impose une valeur minimale en dessous de laquelle la valeur des rayons ne peut descendre (non représenté sur l'algorithme de la figure 3.11). Cette valeur limite est un multiple d'ε, distance de référence définissant les nœuds bordants.

```
BUILD_ADAPTIVE_DD_RRT(q_init)
 1   T.init(q_init);
 2   for k = 1 to K do
 3       repeat
 4           q_rand ← RANDOM_CONFIG();
 5           q_near ← NEAR_NEIGH(q_rand, T, d_near);
 6       until (d_near < q_near.radius)
 7       if q_new ← CONNECT(T, q_rand, q_near)
 8           q_new.radius = ∞;
 9           if q_near.radius ≠ ∞;
10               q_near.radius = (1 + α) × q_near.radius;
11           T.add_vertex(q_new);
12           T.add_edge(q_near, q_new);
13       else
14           if q_near.radius = ∞
15               q_near.radius = R;
16           else
17           q_near.radius = (1 − α) × q_near.radius;
18   Return T;
```

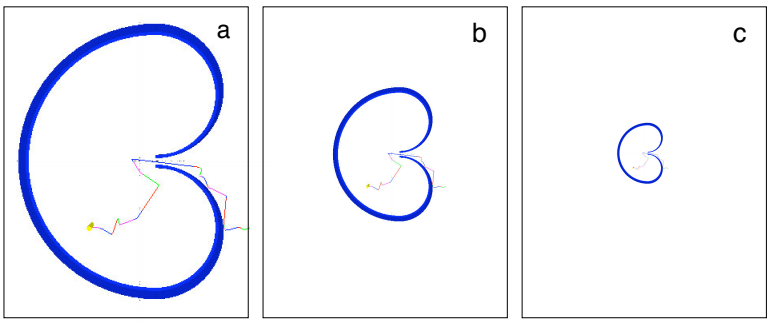
FIG. 3.11 – *Algorithme RRT à Domaine Dynamique avec rayon adaptatif.*

Concernant le choix de α, les résultats de simulation montrent que ce paramètre est beaucoup moins critique que le paramètre de rayon pour l'algorithme DD-RRT initial. Ce paramètre peut donc rester interne au planificateur et demeurer fixe d'une requête à l'autre. Une faible valeur de α (e.g. 2-5%) est suffisante pour améliorer la robustesse de l'algorithme. C'est cet ordre de valeur qui a été utilisé au cours des expérimentations.

L'algorithme est décrit dans sa version monodirectionnelle. Son adaptation à la recherche bidirectionnelle est similaire à celle des RRT classiques (c.f. [Kuffner 00]).

4 Résultats

Comme pour les tests associés au planificateur à environnement changeant, les algorithmes RRT, DD-RRT et DD-RRT$_{adapt}$ ont été implémentés sur la plate-forme Move3D et les simulations réalisées sur une Sunblade 100, 333MHz. Toutes les valeurs rapportées correspondent à des valeurs moyennes calculées pour un ensemble de 50 tests. La première partie présente les performances de l'algorithme dans sa version à rayon fixe alors que la seconde évalue les performances de la version adaptative.



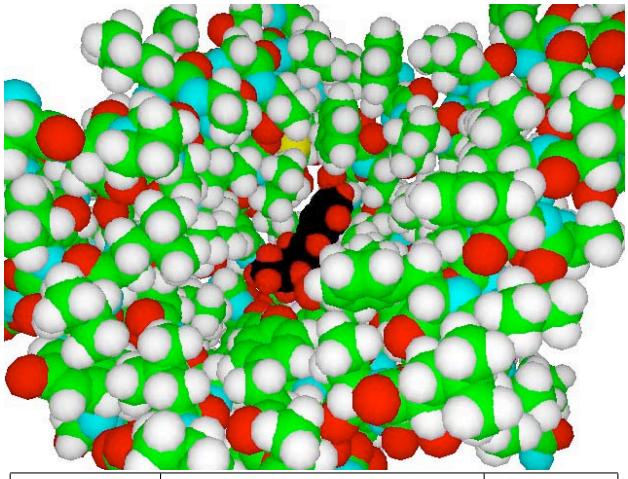
		dynamic-domain bi-RRT	bi-RRT
a	temps (s)	0.4	0.1
	nb. nœuds	253	37
	tests col.	618	54
b	temps (s)	2.5	379
	nb. nœuds	1607	6924
	tests col.	3751	781530
c	temps (s)	1.6	$>$ 80000
	nb. nœuds	1301	–
	tests col.	3022	–

FIG. 3.12 – *Le but dans cet exemple est d'extraire un robot cylindrique hors de la forme géométrique nommée bug trap. Les résultats associés à différentes tailles d'environnements (a-b-c) sont présentés.*

4.1 DD-RRT versus RRT

Nous comparons ici les performances des algorithmes RRT et DD-RRT pour des variantes bidirectionnelles équilibrées. Pour chaque expérimentation, nous indiquons le temps de calcul, le nombre de nœuds de l'arbre solution ainsi que le nombre d'appels au détecteur de collisions réalisés au cours du processus de construction. La valeur du rayon définissant le domaine dynamique est ici fixé à $R = 10\varepsilon$ (ε, distance élémentaire à partir de laquelle on considère qu'un nœud est bordant).

Les premiers résultats concernent l'exemple introductif du bug trap (figure 3.12) pour 3 tailles de l'environnement (a, b ou c). Chaque nouvel environnement possède une surface 50 fois supérieure à celle de l'environnement précédent. Les tests effectués mettent en évidence une rapide détérioration de la performances de l'algorithme RRT usuel lorsque la taille de l'environnement augmente. Alors que le temps moyen est de 0.1 seconde pour (a), il passe à plus de 6 minutes pour (b) et l'environnement (c) n'a pu être résolu en 22 heures de calcul. Comparativement, le temps moyen de résolution pour l'algorithme DD-RRT reste de l'ordre



	dynamic-domain bi-RRT	bi-RRT
temps (s)	70	2926
nb. nœuds	1358	428
tests col.	47710	1257055

FIG. 3.13 – *Exemple de problème moléculaire.*

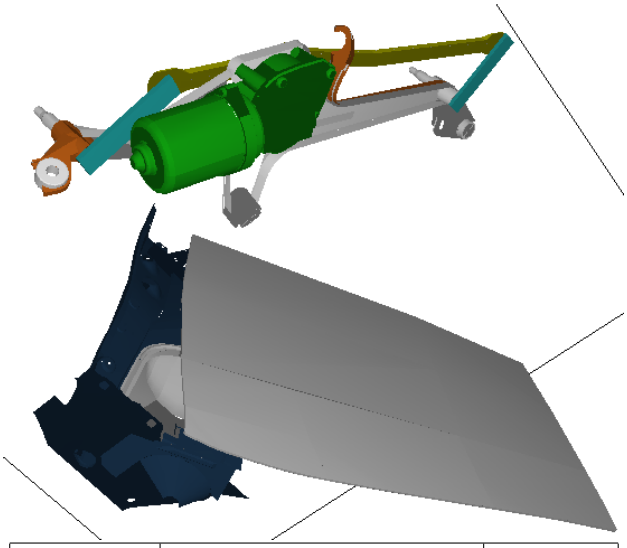
de la seconde dans tous les cas.

Les résultats suivants sont associés au problème moléculaire illustré figure 3.13. Ici, la molécule concernée est modélisée par un corps 3D rigide, son espace des configurations est donc de dimension six. Le but est de trouver un chemin pour un ligand (la petite molécule en noir) permettant d'aller jusqu'au site actif d'une protéine. La principale difficulté provient ici du fait que les tests de collisions sur ce modèle sont très coûteux. L'algorithme RRT réalise davantage de tentatives d'extension infructueuses que l'algorithme DD-RRT, c'est pourquoi il nécessite un temps de résolution plus important. Il est intéressant de noter que malgré le temps de résolution beaucoup plus court, l'algorithme DD-RRT parvient à insérer nettement plus de nœuds, grâce à son domaine d'échantillonnage performant.

Cet exemple moléculaire a également été utilisé pour montrer la capacité de l'algorithme DD-RRT à explorer des espaces de dimension élevée, dans le cas où la molécule est modélisée par une chaîne articulée ayant plus de 300 ddls.

L'exemple de la figure 3.14 est un benchmark de l'industrie automobile[2] [Ferré 04]. Ici, le problème est de vérifier la faisabilité du désassemblage d'un moteur à essuie-glace de son

[2]Nous remercions la compagnie Kineo (www.kineocam.com) pour avoir autorisé la publication de ces résultats obtenus à partir d'un benchmark industriel.



	dynamic-domain bi-RRT	bi-RRT
temps (s)	217	> 80000
nb. nœuds	219	–
tests col.	30443	–

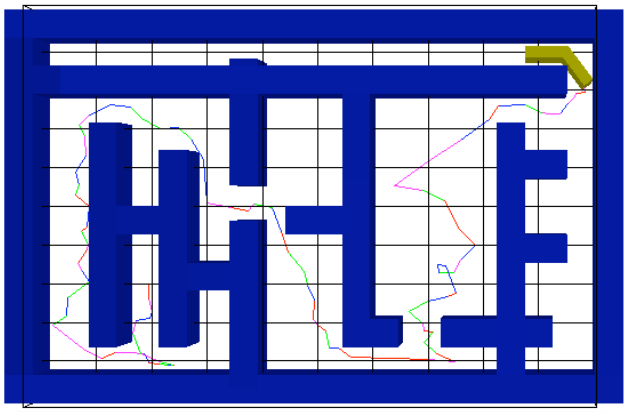
FIG. 3.14 – *Désassemblage du moteur d'essuie-glace d'une carrosserie de voiture.*

support situé sous le capot de la voiture. Ce problème est extrêmement contraint et le chemin solution reste quasiment en contact avec les obstacles. Il s'agit donc d'un benchmark difficile que l'algorithme RRT ne peut résoudre même après plusieurs jours de recherche. L'algorithme DD-RRT résout ce problème avec un temps moyen de quelques minutes.

Ces tests montrent qu'en général, l'algorithme DD-RRT apporte un gain de performance significatif comparé à RRT, en particulier dans les problèmes très contraints. Dans certains cas, le gain peut cependant être moins important. La figure 3.15 montre un exemple bidimensionnel avec un robot articulé à 4 ddls. Le but est de déplacer ce robot d'un coin du labyrinthe à un autre. Cet exemple correspond au problème de l'exploration d'un labyrinthe contrairement aux exemples précédents qui correspondent à des problèmes de désassemblage contraints. De plus, la géométrie de la scène 2D est extrêmement simple. Même pour ce type de problème favorable à RRT, DD-RRT reste légèrement plus performant.

4.2 DD-RRT$_{adapt}$ versus DD-RRT

Dans cette section, nous comparons plus particulièrement la version DD-RRT adaptative à la version DD-RRT à rayon fixe utilisée pour les tests précédents. Comme précédemment, les

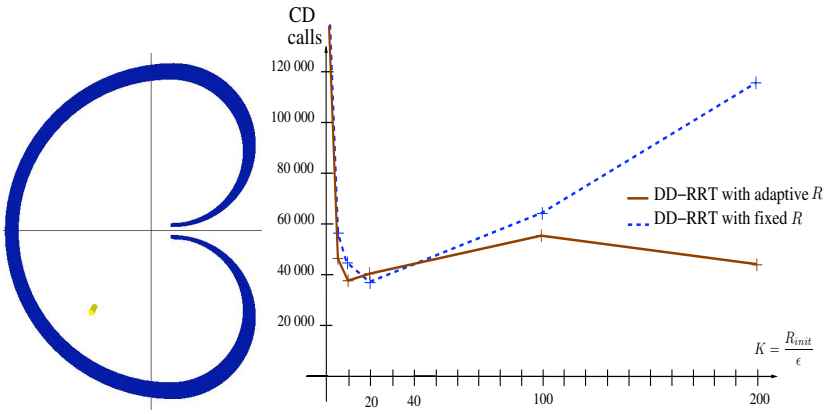


	dynamic-domain bi-RRT	bi-RRT
temps (s)	161	237
nb. nœuds	25483	20392
tests col.	604503	464137

FIG. 3.15 – *Le but dans cet exemple est de déplacer un corps articulé à 4 degrés de liberté d'un coin du labyrinthe à l'autre.*

valeurs initiales des rayons R sont définies à partir de la distance élémentaire ε : $R = K\varepsilon$. Les courbes associées aux figures 3.16, 3.17 et 3.18, montrent l'évolution des performances (en terme de nombre d'appels au détecteur de collisions) en fonction de la valeur du paramètre K. Les tableaux indiquent pour chaque exemple et pour chaque valeur de K, les temps de calcul, le nombre de nœuds développés, ainsi que le nombre d'appels au détecteur de collisions pour la construction de l'arbre.

Les résultats indiqués dans la figure 3.16 reprennent l'exemple de référence bugtrap dans sa version difficile où la taille de l'environnement représente 150 fois celle du bug trap. Pour cet exemple, la valeur du rayon optimal est proche de $R = 20\varepsilon$. Pour cette valeur, le nombre d'appels au détecteur de collisions est plus de 40 fois plus faible que pour l'algorithme RRT classique ! Pour un rayon inférieur, les performances diminuent assez rapidement. Lorsque le rayon est supérieur à l'optimal, elles diminuent également mais la pente associée est plus douce. Avec la version adaptative, on peut noter que la performance de l'algorithme reste stable quand la valeur est surestimée et est seulement affectée par de trop faibles rayons. Notons également que pour cet exemple, les nœuds frontières sont tous contraints par les obstacles de façon très semblable. Ainsi la bonne performance de la version adaptative provient ici davantage du meilleur équilibrage entre exploration et affinage plutôt que d'une différentiation locale des valeurs des rayons.



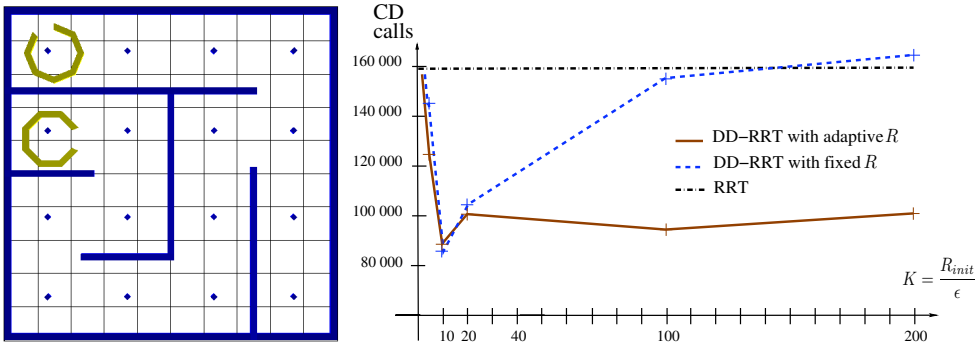
	Rayon adaptatif					RRT
K	5	10	20	100	200	
temps (s)	14	12	14	28	38	719
nb. nœuds	6 263	4 992	5 134	5 929	4 909	6 929
tests col.	47 440	38 711	40 869	51 300	44 832	1 627 974

	Rayon fixe				
K	5	10	20	100	200
temps (s)	17	15	13	28	66
nb. nœuds	7 522	5 868	4 665	4 638	5 136
tests col.	56 900	45 596	38 229	59 384	115 214

FIG. 3.16 – *Influence de la version adaptative sur les performances pour l'exemple canonique du bug trap.*

L'exemple suivant (figure 3.17) montre un robot en rotation/translation dans un environnement 2D de type labyrinthe. La difficulté du problème vient de la nécessité de combiner translations et rotations pour se déplacer dans le labyrinthe. L'exemple est peu favorable à l'algorithme DD-RRT : le problème est principalement exploratoire et la complexité géométrique de la scène est faible. Néanmoins en terme de tests de collisions, cette version permet encore d'être très robuste vis à vis de la valeur du rayon initial, même si pour cette situation, le temps total de résolution est ici légèrement supérieur pour la version adaptative.

Le dernier exemple (figure 3.18) met en jeu un bras manipulateur à 6 degrés de liberté saisissant un cerceau qui doit être extrait d'une barre en forme de S. Avec la méthode non adaptative, l'efficacité de l'algorithme DD-RRT décroît très rapidement quand l'écart entre le rayon fixé et le rayon optimal grandit. Quand celui-ci est égal à 10 fois la valeur optimale ($K = 100$ au lieu de 10), le gain en terme d'appels au détecteur de collisions chute à moins de 40%. Pour la version adaptative et pour le même écart, le gain reste d'un facteur six indépendamment du rayon choisi.



	Adaptive radius					RRT
K	5	10	20	100	200	
temps (s)	206	157	179	148	144	96
nb. nœuds	22 190	11 932	12 160	9 186	9 783	10 419
tests col.	125 233	88 781	103 512	94 139	102 609	159 392

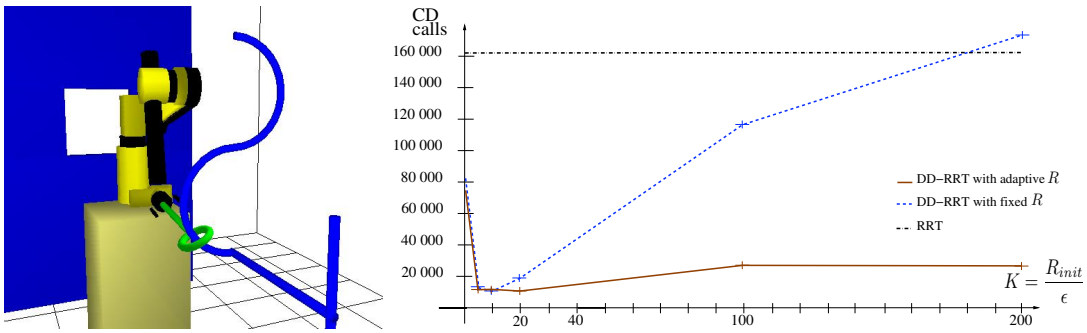
	Fixed radius				
K	5	10	20	100	200
temps (s)	124	58	68	93	95
nb. nœuds	22 685	9 604	9 503	10 524	10 678
tests col.	144 197	86 482	104 953	154 201	165 035

FIG. 3.17 – *Environnement de type labyrinthe. Le robot en forme de C doit atteindre l'extrémité opposée du labyrinthe en combinant habilement rotations et translations.*

Ces résultats montrent que, la méthode adaptative permet de fortement augmenter la robustesse de l'algorithme vis à vis du paramètre de rayon qui contrôle les domaines dynamiques. Le gain est particulièrement significatif quand la valeur initiale est une surestimation de la valeur optimale plutôt que l'inverse. Grâce à cette propriété, notre algorithme dans sa version adaptative peut être initialisé arbitrairement par un rayon initial important, ce qui permet d'éliminer la dépendance du choix de ce paramètre sur la performance de l'algorithme.

4.3 Conclusion

Ces simulations montrent de façon générale une performance meilleure pour l'algorithme DD-RRT que pour l'algorithme RRT. Ce gain est particulièrement vrai dans des problèmes contraints localement et des géométries complexes. L'algorithme est moins efficace dans les problèmes d'exploration où le chemin solution est long, comme pour le cas des labyrinthes. La méthode proposée est donc tout à fait adaptée pour la reconnexion dynamique d'arêtes de l'approche décrite au chapitre 2 : les problèmes de reconnexion sont des problèmes localement contraints pour lesquels il n'est pas nécessaire d'explorer une grande portion de l'espace.



	Rayon adaptatif					RRT
K	5	10	20	100	200	
temps (s)	34	24	21	75	80	240
nb. nœuds	1 029	1 009	718	943	844	1 314
tests col.	11 166	11 160	11 468	27 786	27 343	162 570

	Rayon fixe				
K	5	10	20	100	200
temps (s)	47	21	28	166	245
nb. nœuds	1 277	890	724	809	856
tests col.	12 408	10 871	18 107	117 903	174 714

FIG. 3.18 – *Résultats pour un bras manipulateur à 6 ddls. Celui-ci manipule un cerceau qui doit être extrait d'une barre en forme de S.*

Chapitre 4

Réseaux de Rétraction

La méthode de mise à jour dynamique de réseaux probabilistes décrite au chapitre 2 repose sur le calcul préalable d'un réseau rendant compte des contraintes imposées par les obstacles statiques de la scène. La performance globale de cette approche repose sur la construction d'un réseau suffisamment riche, c'est a dire contenant des solutions alternatives lorsque certaines portions sont invalidées par les obstacles mobiles. L'existence de solutions alternatives dans le réseau permet en effet d'éviter la mise à jour dynamique d'arêtes invalides par la méthode de diffusion décrite au chapitre 3, qui malgré sa performance reste l'étape la plus coûteuse de notre approche. Le réseau doit donc être suffisamment riche, mais sans pour autant avoir une taille inutilement grande afin d'éviter un coût de construction trop élevé et une dégradation de la performance des requêtes de planification.

Dans ce chapitre, nous proposons une nouvelle forme de réseaux probabilistes, appelés *réseaux de rétraction*, dont l'intérêt est de combiner ces deux critères pourtant antagonistes. La première section introduit ces critères en établissant un lien avec la notion d'homotopie, qui est une propriété plus large que la propriété de connexité généralement considérée dans les réseaux probabilistes. Nous formalisons ensuite cette notion de réseau de rétraction avant de présenter une méthode générale, basée sur une extension de l'algorithme Visib-PRM à des graphes faiblement redondants. Les résultats indiqués en fin de chapitre montrent la capacité de cette méthode à calculer efficacement des réseaux adaptés à notre problème, c'est à dire contenant les cycles utiles au calcul de solutions alternatives lors de requêtes de planification.

1 Connexité versus homotopie

Quelles sont les caractéristiques souhaitées pour un réseau représentatif de l'espace libre ? Deux propriétés (définies plus formellement dans l'annexe A) sont avant tout requises. D'une part, les nœuds et les arêtes du réseau doivent capturer la *couverture* et la *connexité* de $\mathcal{C}_{free}$. Ce premier critère est généralement rempli par les méthodes PRM existantes et ils constituent une condition nécessaire à leur complétude probabiliste. D'autre part, un autre critère, plus riche que la connexité, concerne la capture des classes d'homotopie. Une définition plus formelle de l'homotopie est donnée en annexe A. Dans cette section, nous nous limitons à une présentation intuitive de cette propriété à travers un exemple illustré par la figure 4.1. Dans cet exemple, l'espace des configurations du robot est composé de deux classes d'homotopie qui se traduisent en deux classes de chemins susceptibles de relier les configurations de la figure de gauche (a). Les figures (b) et (c), montrent l'intérêt d'un réseau basé sur cette notion d'homotopie, qui permet dans cet exemple de tenir compte des portes de l'environnement qui peuvent invalider les chemins d'une classe donnée suivant qu'elles sont ouvertes ou fermées.

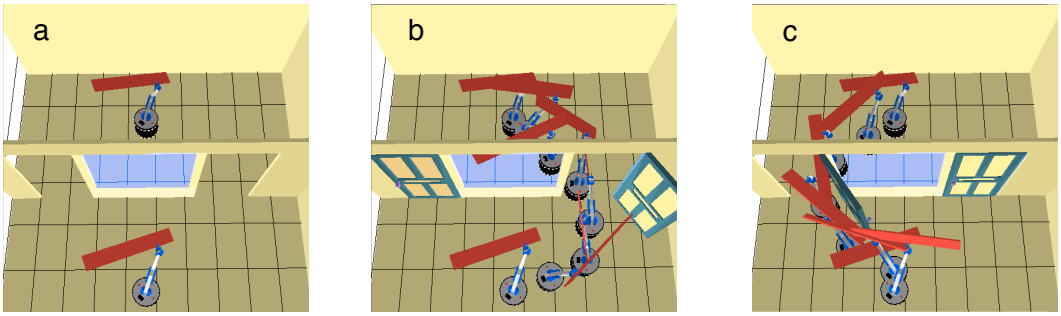


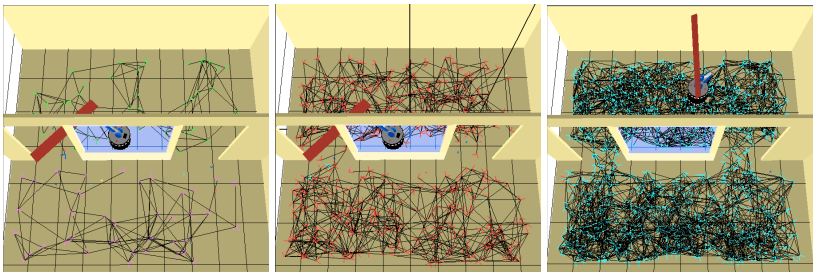
FIG. 4.1 – *Le problème des portes ouvertes/fermées dans un environnement d'intérieur illustre la nécessité de capturer les classes d'homotopie de l'espace libre. Les deux chemins solutions présentés appartiennent ici à deux classes d'homotopie différentes.*

Du point de vue des algorithmes de planification, les méthodes traditionnelles ne garantissent pas forcément cette propriété. En particulier, les réseaux à structure d'arbre, ne possédant donc aucun cycle, ne peuvent représenter qu'une seule classe d'homotopie. C'est le cas notamment de l'algorithme Visib-PRM [Nissoux 99].

De façon générale, plus un réseau contient de cycles, plus il a de chances de capturer les classes d'homotopie. Dans les algorithmes PRM standards, on peut par exemple utiliser une stratégie de connexion correspondant aux k nœuds les plus proches [Kavraki 96]. Une autre stratégie de connexion utilisée dans [Bohlin 00] est basée sur la distance maximale aux nœuds voisins d. Dans les deux cas, les chances de capturer les classes d'homotopie augmentent avec la valeur de k ou de d, mais au prix d'une perte de performance significative sur des problèmes

nécessitant le calcul d'un réseau dense. La difficulté de ces stratégies est aussi qu'il n'existe pas de méthode permettant de déterminer de manière automatique quel couple (N,k) ou (N,d) choisir pour garantir un réseau capturant correctement l'homotopie.

Nous illustrons ces difficultés à travers l'exemple de la figure 4.2. Suivant la densité d'échantillonnage, le réseau calculé peut réussir à capturer zéro (à gauche), un (au centre) ou deux (à droite) types de passages reliant les deux pièces de l'environnement. Le nombre de passages trouvés et le temps de construction associé sont indiqués en fonction du couple (N,k) choisi. On voit ici que seul le couple $(N = 1000, k = 10)$, permet de capturer correctement les deux passages, au dépend d'une structure de réseau extrêmement dense, composée de beaucoup de nœuds et d'arêtes peu utiles.



			N				
			50	100	200	500	1000
k	5	N.b chem.	0.2	0.4	0.5	1.1	1.7
		temps (s)	1	4	9	28	80
	10	N.b chem.	0.9	1.0	1.1	1.5	2.0
		temps (s)	3	6	15	40	106

FIG. 4.2 – *Nombre de chemins connectant les deux pièces et temps de construction en fonction du nombre de plus proches voisins auxquels on essaye de se connecter (k) et du nombre total de nœuds (N).*

Ces problèmes ont été peu traités dans les travaux récents sur la planification de mouvement probabiliste. Seuls les travaux de [Schmitzberger 02] ont abordé le problème sous un angle formel et proposé une méthode permettant de capturer les classes d'homotopie pour un réseau probabiliste peu dense construit à partir de l'algorithme Visib-PRM [Siméon 00]. La construction du réseau comporte trois étapes. La première construit un arbre avec la méthode Visib-PRM. Dans un second temps, cet arbre est enrichi pour obtenir une structure de *réseau connexe d'un point de vue quelconque* (c.f section 3). Enfin, une dernière étape de filtrage des cycles redondants est proposée pour supprimer ceux qui ne participent pas à la formation de nouvelles classes d'homotopie. Bien que le principe de cette approche soit intéressant, il s'agit cependant de résultats préliminaires et les algorithmes proposés se limitent à des cas particuliers de problèmes dans des espaces de faible dimension. En particulier, l'opération la plus

délicate correspond au filtrage des cycles redondants de la troisième étape.

Mais un réseau se limitant aux simples classes d'homotopie constitue-t-il vraiment la structure de données la mieux adaptée pour la planification de mouvements ? Considérons l'exemple de la figure 4.3. L'espace des configurations libres n'est composé que d'une classe d'homotopie. Ainsi, un réseau ne capturant que les classes d'homotopie ne contiendra aucun cycle et possèdera donc une structure d'arbre. Pourtant, même si la "nature topologique" des deux chemins présentés est la même, il existe des différences suffisamment notables pour que l'on souhaite capturer chacun d'eux au sein du réseau. De manière générale, plus la déformation entre deux chemins est une opération compliquée, plus il est intéressant de distinguer leur représentation au sein du réseau.

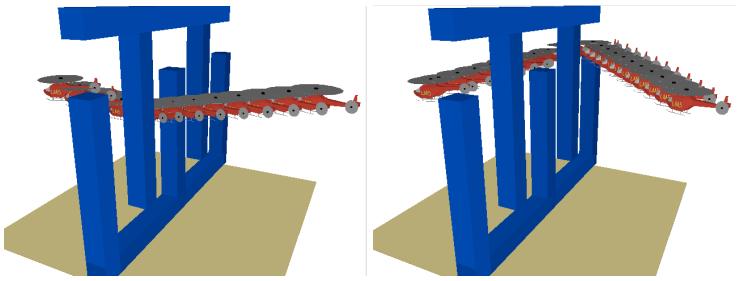


FIG. 4.3 – *Un réseau se limitant aux différentes classes d'homotopie ne permet pas de représenter les deux types de chemins visualisés*

Dans la suite de ce chapitre, nous présentons une nouvelle méthode de construction de réseaux appelés réseaux de rétractions. Cette méthode prend comme point de départ les travaux proposés dans [Schmitzberger 02], notamment à travers la notion de réseau connexe d'un point de vue quelconque. Elle utilise également un nouveau type d'opérateur géométrique qui permet de définir des *opérations de rétraction* entre chemins. Au final, les algorithmes présentés permettent de construire des réseaux de taille réduite vérifiant une propriété proche de l'homotopie mais mieux adaptée à la planification de mouvements. Ces réseaux sont calculés à travers une opération de rétraction qui caractérise les classes de chemins qu'il est intéressant de capturer dans la structure de graphe.

2 Réseaux de rétraction

Dans cette section, nous définissons la notion de *diagramme de Visibilité* de chemins à partir duquel se définit l'opération de rétraction et introduisons ensuite la notion de réseau de rétraction.

2.1 Diagramme de Visibilité de chemins

Soient τ, τ', deux chemins de $\mathcal{C}_{free}$ et soient des paramétrisations $t,t' : [0,1] \rightarrow \mathcal{C}_{free}$ associées respectivement à chacun des deux chemins. Un couple de paramètres (t, t') correspond alors à un unique couple de configurations $(q_t, q_{t'})$ situées respectivement sur τ et τ'.

A partir de la méthode locale linéaire $\mathcal{L}$, on définit la fonction de *visibilité paramétrée* $Vis : [0,1] \times [0,1] \rightarrow \{0,1\}$ telle que $Vis(t,t') = 1$ si $\mathcal{L}(\tau(t), \tau'(t')) \in \mathcal{C}_{free}$ et $Vis(t,t') = 0$ sinon (c.f. figure 4.4).

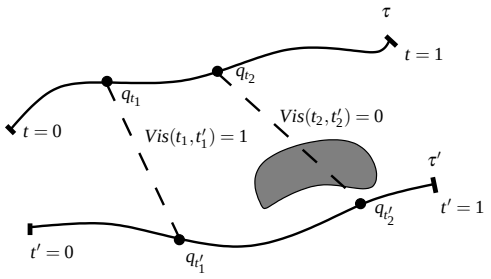


FIG. 4.4 – *Fonction de visibilité paramétrée*

On appelle *diagramme de visibilité* le graphe de dimension deux, associé à la fonction de visibilité paramétrée (c.f figure 4.5).

2.2 Rétraction par visibilité

Les chemins τ et τ' sont dits mutuellement *rétractables par visibilité* si pour le *diagramme de visibilité* associé à ces deux chemins, il existe un chemin reliant le point de coordonné $(0,0)$ au point de paramétrisation $(1,1)$.
Autrement dit, τ et τ' sont mutuellement *rétractables par visibilité* s'il existe une fonction paramétrée continue f, définie par :

$$f : \ [0,m] \ \rightarrow \ [0,1]^2$$

et telle que :

$$\begin{cases} f(0) & = & (0,0) \\ f(1) & = & (1,1) \\ Vis(f(s)) & = & 1, \forall s \in [0,m] \end{cases}$$

De manière plus informelle, deux chemins sont mutuellement *rétractables par visibilité* lorsqu'il est possible de les parcourir tout en satisfaisant une contrainte de visibilité permettant de passer de l'un à l'autre par un chemin local. Par la suite et pour simplifier l'écriture, on parlera simplement de "rétraction" et de chemin "rétractable".

Des exemples de diagrammes de visibilité pour des couples de chemins ayant leurs positions extrémales confondues sont présentés figure 4.5. Seuls les chemins du dernier scénario (d) sont mutuellement rétractables, car il existe un chemin reliant les points (0,0) et (1,1) dans le diagramme de visibilité.

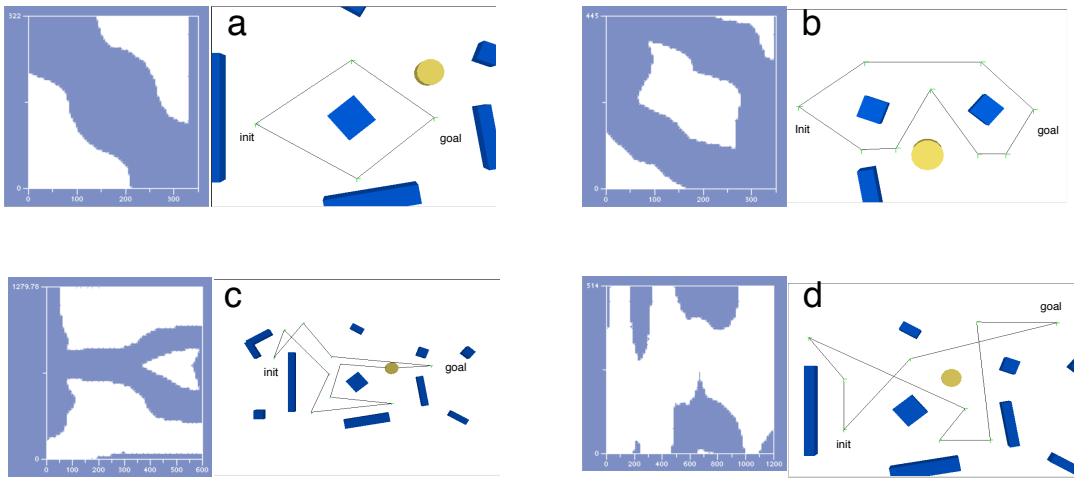


FIG. 4.5 – *Exemples de diagrammes de visibilité pour des paires de chemins ayant leurs configurations extrémales confondues. Les zones blanches correspondent aux couples de paramètres pour lesquels il y a visibilité. L'opération de rétraction par visibilité n'est ici possible que pour le dernier exemple (d), car il existe un chemin dans le diagramme de visibilité reliant les points* $(0,0)$ *et* $(1,1)$.

2.3 Réseaux de rétraction

L'outil de rétraction précédemment présenté nous permet de définir la notion de réseau de rétraction par visibilité, appelé par la suite réseau de rétraction :

Définition 2.1 *G est un réseau de rétraction si et seulement si pour tout chemin valide* τ *de* $\mathcal{C}_{free}$*, il existe au moins un chemin issu de G sur lequel* τ *est rétractable par visibilité.*

2.4 Retraction versus homotopie

Nous présentons dans cette section le lien entre rétraction et homotopie. Ceci permet de préciser les caractéristiques des réseaux de rétraction.

La relation de rétraction liant deux chemins n'est pas une relation d'équivalence contrairement à la relation d'homotopie. En effet la relation est non transitive. Cette propriété provient

directement de la non transitivité de la relation de visibilité (c.f figure 4.6). Pour cette raison, la rétraction ne peut servir à caractériser la nature topologique d'un espace des configurations, contrairement à l'homotopie.

FIG. 4.6 – *La relation de rétraction est non transitive : τ est rétractable sur τ' et τ' sur τ'', mais τ n'est pas rétractable sur τ''*

Par contre, deux chemins mutuellement rétractables font partie de la même classe d'homotopie. En effet, le chemin solution du diagramme de visibilité décrit une transformation continue particulière entre les deux chemins. Grâce à cette propriété, la structure des réseaux de rétraction englobe les classes d'homotopie : tout chemin de l'espace peut être rétracté sur le réseau, donc peut être déformé en un chemin du réseau. Les réseaux de rétractions possèdent donc une structure plus riche que les classes d'homotopie et donc mieux adaptée pour les problèmes de planification de mouvement.

Nous décrirons par la suite une méthode permettant de construire des réseaux de rétraction de taille réduite. Pour cela, il est préalablement nécessaire d'introduire la notion de réseaux connexes d'un point de vue quelconque, qui fait l'objet de la section suivante.

3 Réseau connexe d'un point de vue quelconque

Le formalisme des réseaux connexes d'un point de vue quelconque a été introduit dans [Schmitzberger 02]. Nous reprenons ici cette notion et établissons la connexion avec les réseaux de rétraction.

3.1 Sous réseau visible

Soient $\mathcal{L}$ une méthode locale donnée, un réseau $G = (N,E)$ et une configuration q_v. Si G représente une couverture de $\mathcal{C}_{free}$, on peut extraire de N un ensemble minimal de nœuds gardiens N_g assurant cette couverture. Les autres nœuds sont appelés connecteurs. On peut alors définir le sous réseau visible $G_v = (N_v, E_v)$ associé à la configuration q_v tel que :

– N_v, sous liste de nœuds gardiens visibles depuis la configuration q_v :

$$N_v = \{n \in N_g / \mathcal{L}(q_v, q(n)) \in \mathcal{C}_{free}\}$$

– E_v, sous liste d'arêtes visibles depuis la configuration q_v :

$$E_v = \{e \in E / \mathcal{L}(q_v, e) \in \mathcal{C}_{free}\}$$

Avec pour notation $\mathcal{L}(q_v, e) \in \mathcal{C}_{free}$ si $\{\forall q \in e, \mathcal{L}(q_v, q) \in \mathcal{C}_{free}\}$.

Des exemples de sous réseaux visibles sont présentés figure 4.7.

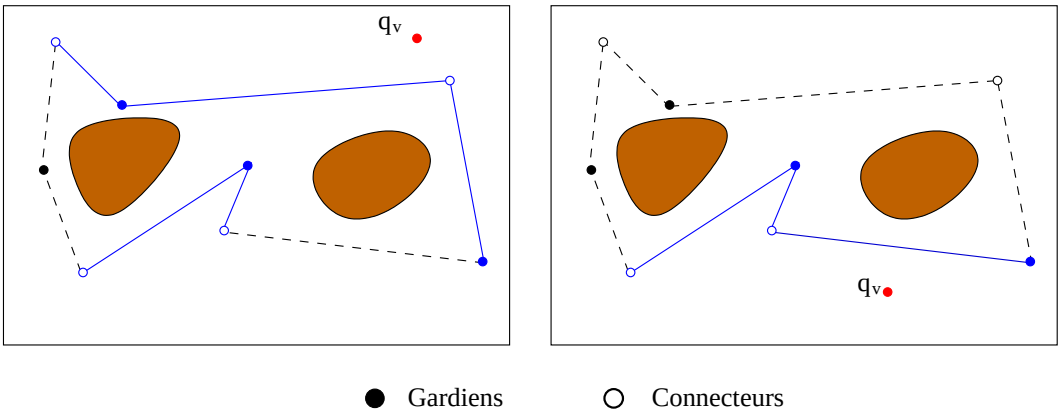


FIG. 4.7 – *Deux exemples de sous réseaux visibles à partir d'une configuration q_v donnée. A gauche, celui-ci n'est pas connexe du point de vue de q_v alors qu'il l'est à droite.*

Remarque : On supposera que G ne comporte qu'une seule composante connexe. Dans le cas contraire, on considère de façon indépendante le sous réseau visible pour chacune des composantes connexes.

3.2 Réseau connexe

Définition 3.1 *Un réseau connexe d'un point de vue quelconque est un réseau tel que pour toute configuration de l'espace, le sous réseau visible associé est connexe.*

Nous présentons maintenant une propriété très importante qui va servir de support à notre algorithme de construction de réseaux de rétraction.

Lien avec les réseaux de rétraction : Un réseau connexe d'un point de vue quelconque est un cas particulier de réseau de rétraction.

Preuve : Soit G, un réseau connexe d'un point de vue quelconque et soit τ un chemin donné de $\mathcal{C}_{free}$. τ peut être partitionné en $2n-1$ sous chemins successifs :

$$\tau = \{\tau_{g_1} \oplus \tau_{g_1 \cap g_2} \oplus \ldots \oplus \tau_{g_i} \oplus \tau_{g_i \cap g_{i+1}} \oplus \tau_{g_{i+1}} \oplus \ldots \tau_{g_{n-1}} \oplus \tau_{g_{n-1} \cap g_n} \oplus \tau_{g_n}\}$$

Avec τ_{g_i} portion de chemin visible uniquement par le gardien g_i et $\tau_{g_i \cap g_{i+1}}$ portion visible conjointement par les gardiens g_i et g_{i+1} (c.f. figure 4.8).

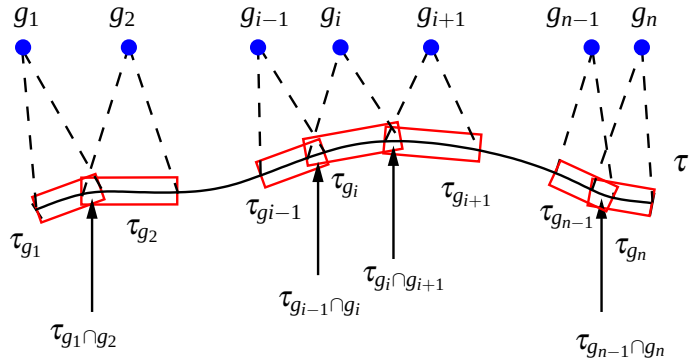


FIG. 4.8 – *Décomposition d'un chemin en fonction des zones de visibilité des nœuds gardiens du réseau.*

La portion de chemin τ_{g_i} est rétractable sur g_i et la portion $\tau_{g_{i+1}}$ sur g_{i+1}. Le fait que G soit connexe d'un point de vue quelconque assure, pour chaque configuration q_v du chemin située à l'intersection du domaine de visibilité de deux gardiens (*i.e.* $\in \tau_{g_i \cap g_{i+1}}$), l'existence d'un chemin du réseau visible de cette configuration et reliant ces deux gardiens (c.f. figure 4.9). Ainsi, par construction, on s'assure de pouvoir rétracter le chemin en question sur un chemin du réseau.

Notons que pour deux configurations de τ appartenant au même intervalle $\tau_{g_i \cap g_{i+1}}$, les chemins visibles du réseau connectant les deux gardiens ne sont pas nécessairement les mêmes (distinction entre traits pleins et traits pointillés figure 4.9).

Les réseaux visibles d'un point de vue quelconque sont une forme particulière de réseaux de rétraction. Ces réseaux sont cependant encore inutilement redondants. Nous allons voir au cours de la section suivante qu'il est possible de filtrer certains chemins de ces réseaux, tout en conservant la propriété de réseaux de rétraction.

3.3 Chemins redondants

Considérons un réseau visible d'un point de vue quelconque. Il existe alors un ensemble minimal N_g, de nœuds gardiens assurant la couverture de l'espace. Considérons alors g_a et g_b deux nœuds gardiens appartenant à N_g et γ, γ', deux chemins du réseau reliant ces deux gardiens et formant donc un cycle. On a alors la propriété suivante :

Propriété : Si γ et γ' sont mutuellement rétractables, on peut supprimer un des deux chemins tout en conservant la propriété de réseau de rétraction.

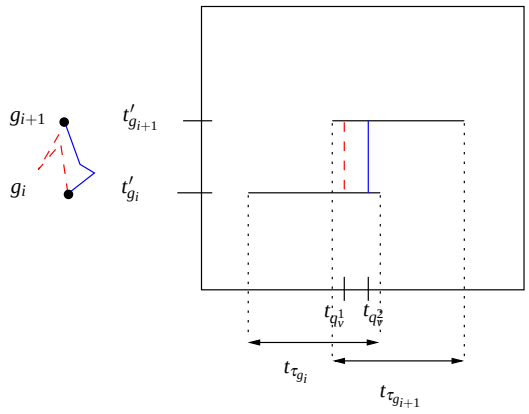


FIG. 4.9 – *Existence d'un chemin dans le diagramme de visibilité pour un réseau connexe d'un point de vue quelconque : pour toute configuration située à l'intersection du domaine de visibilité de deux gardiens ($\in \tau_{g_i \cap g_{i+1}}$), il existe un chemin visible du réseau reliant ces deux gardiens.*

Preuve : La démonstration de cette propriété se rapproche de celle assurant la propriété de rétraction pour les réseaux visibles d'un point de vue quelconque. Supposons un réseau visible de tout point de vue dont on a enlevé les chemins γ rétractables sur des chemins γ' déjà présents dans le réseau. Considérons également un chemin τ de l'espace et la partition de ce chemin telle qu'on l'a déjà définie figure 4.8. Pour les mêmes raisons que précédemment, les configurations appartenant à τ_{g_i} sont rétractables sur g_i, et celles de $\tau_{g_{i+1}}$ sur g_{i+1}. Pour chaque configuration q_v de $\tau_{g_i \cap g_{i+1}}$, deux cas peuvent se présenter. Soit q_v voit un chemin du réseau reliant g_i à g_{i+1} (i.e. g_i et g_{i+1} sont connexes du point de vue de q_v), soit le réseau n'est pas connexe du point de vue de q_v. Dans ce cas, il existe alors un chemin $\{g_i \rightarrow q_v \rightarrow g_{i+1}\}$ rétractable sur le réseau. Au niveau du diagramme de visibilité, cela signifie que pour le paramètre t_{q_v}, il existe un chemin du réseau reliant les paramètres t'_{g_i} et $t'_{g_{i+1}}$ (c.f. figure 4.10).

La propriété précédente permet d'obtenir un critère pour supprimer des chemins d'un réseau connexe d'un point de vue quelconque tout en conservant la propriété de réseau de rétraction. Au cours de la section suivante, nous décrirons un algorithme de construction de réseaux de rétraction, qui s'appuie sur ce résultat pour conserver des réseaux de taille minimale.

4 Réseau de rétraction : Construction

4.1 Principe général

L'algorithme général de construction des réseaux de rétraction Retract_PRM est présenté figure 4.11. D'abord, un premier arbre est construit à l'aide de l'algorithme Visib-PRM. Ensuite,

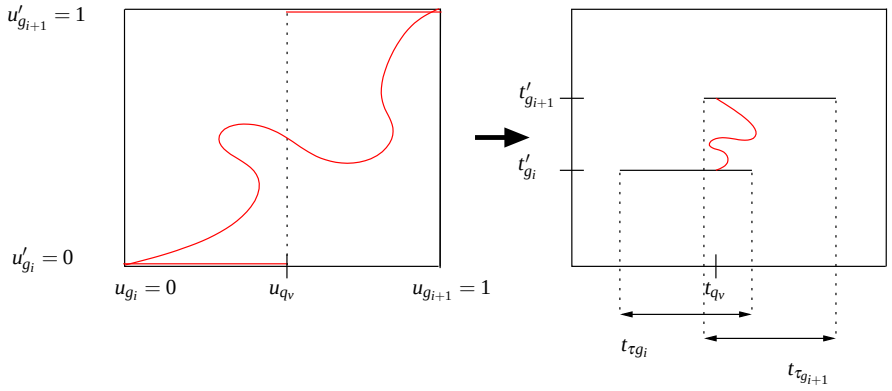


FIG. 4.10 – *Existence d'un chemin dans le diagramme de visibilité en considérant les rétractions. Pour une configuration q_v de $\tau_{g_i \cap g_{i+1}}$, s'il n'y a pas de chemin visible reliant g_i à g_{i+1}, alors il existe un chemin $\{g_i \rightarrow q_v \rightarrow g_{i+1}\}$ rétractable sur un chemin du réseau reliant g_i à g_{i+1}. Ceci assure au niveau du diagramme de visibilité l'existence d'un chemin reliant les paramètres t'_{g_i} et $t'_{g_{i+1}}$.*

à chaque itération, une configuration libre q_v est tirée aléatoirement et la connexité du sous réseau visible est calculée (fonction *TestVisibConnection* ligne 5). L'évaluation de la connexité se fait en évitant si possible, le calcul intégral de ce sous réseau. Lorsque le sous réseau visible n'est pas connexe, avant d'insérer q_v dans le réseau, on cherche à savoir si celui-ci ne va pas participer à la création d'un chemin redondant tel que nous l'avons défini section 3.3. Pour cela, on détermine deux composantes distinctes quelconques du sous réseau et parmi celles-ci, on choisit les nœuds n_1, n_2 plus proches voisins de q_v. Ensuite, on teste l'éventuelle rétraction du chemin $\tau = n_1 - q_v - n_2$ sur des chemins du réseau (fonction *TestRetract* ligne 9). Si la rétraction est possible, on rejette la configuration car celle-ci n'est pas nécessaire à la construction du réseau de rétraction.

La méthode permettant de déterminer la connexité d'un sous réseau visible (fonction *TestVisibConnection*), ainsi que la façon dont est testée la rétraction (fonction *TestRetract*) font l'objet des deux sous sections suivantes.

4.2 Sous réseau visible

Nous présentons ici la méthode de calcul de la connexité d'un sous réseau visible à partir d'une configuration q_v (fonction *TestVisibConnection* de l'algorithme Retract_PRM). L'algorithme associé à cette recherche est représenté figure 4.12. Dans un premier temps, on détermine l'ensemble des nœuds du sous réseau visible en testant à l'aide de la méthode locale les chemins reliant q_v aux différents nœuds du réseau. Ensuite, on examine la connexité des nœuds visibles. Cet examen se fait en deux passes. Dans un premier temps, on considère

RETRACT_PRM
input : the robot A, the environment B, $ntry_{max}$, $ntry\_cycl_{max}$
output : the roadmap G

```
1   G ← Visib-PRM(A, B, ntry_max)
2   ntry ← 0
3   While ntry < ntry_cycl_max
4       q_v ← RandomFreeConfig(A, B)
5       If TestVisibConnection = False
6           n_1 ← NearestNode(q_v, Conn_1(G_v))
7           n_2 ← NearestNode(q_v, Conn_2(G_v))
8           τ ← BuildPath(n_1, q_v, n_2)
9           If TestRetract(τ, n_1, n_2, G) = 0
10              AddNodeAndEdges(q_v, n_1, n_2, G)
11              n_try ← 0
12          Else
13              n_try ← n_try + 1
14          End If
15      Else
16          n_try ← n_try + 1
17      End If
18  End While
```

FIG. 4.11 – *Algorithme globale de génération d'un réseau de rétraction*

toutes les arêtes potentiellement visibles. Ainsi deux nœuds sont détectés non connexes s'il est nécessaire de passer par un nœud invisible pour les relier (fonction *IsVisibleConnect* ligne 8). Ce premier test permet d'augmenter l'efficacité de la méthode en éliminant des sous réseaux non-connexes avant même d'avoir fait des tests de visibilité d'arêtes qui sont plus coûteux. Si ce premier test ne permet pas de déterminer la connexité du réseau, on détermine cette fois-ci la connexité en testant la visibilité des arêtes séparant les nœuds en question (fonction *IsVisibleConnect* ligne 12).

Nous décrivons maintenant le test de visibilité d'une arête à partir d'une configuration donnée.

Visibilité d'une arête

Pour le cas le plus simple, le test de visibilité d'une arête se ramène au test de validité d'une facette dans l'espace des configurations, ayant pour sommets q_v et les deux extrémités de l'arête (figure 4.13).

En réalité, ce test de visibilité peut se décomposer en plusieurs tests élémentaires de facettes, suivant la nature de l'espace des configurations :

– Dans le cas où $\mathcal{C}$ est isomorphe à $[0,1]^n$ (les degrés de liberté du robot sont uniquement des translations ou des rotations bornées), alors le test de visibilité d'une arête se ramène

```
TestVisibConnection(G, q_v)
1   N_vis ← EmptyList
2   For all node n ∈ N_g
3       If Visible(n, q_v)
4           AddToList(n, N_vis)
5       End If
6   Endfor
7   TestEdges ← False
8   If IsVisibleConnect(q_v, N_vis, G,TestEdges) = False
9       Return False
10  End If
11  TestEdges ← True
12  If IsVisibleConnect(q_v, N_vis, G ,TestEdges) = False
13      Return False
14  End If
15  Return True
```

FIG. 4.12 – *Algorithme de test de la connexité du réseau visible de q_v*

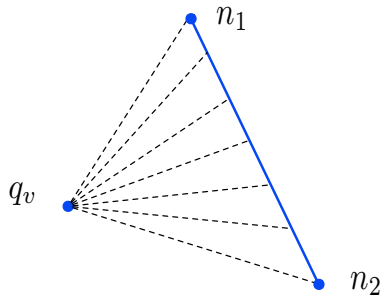


FIG. 4.13 – *La visibilité d'une arête peut s'exprimer par rapport à la validité d'une facette de $\mathcal{C}$.*

effectivement au test élémentaire de la facette de sommets q_v et les deux extrémités de l'arête (c.f. figure 4.14(a)).

– Dans le cas plus complexe où un ou plusieurs degrés de liberté sont circulaires, (espace isomorphe à $[0,1]^n \times SO(d)^m$, avec $m > 0$), le test de visibilité de l'arête se décompose en plusieurs tests élémentaires de facettes (c.f. figure 4.14(b), (c), (d)). Pour déterminer ces facettes, on vérifie pour chaque degré de liberté s'il n'y a pas de changement d'orientation de la visibilité quand on parcourt l'arête. Ce changement se produit quand la distance entre q_v et une configuration donnée de l'arête est égale à π pour la projection suivant le degré de liberté (i.e. les deux configurations sont diamétralement opposées vis à vis de ce degré de liberté). Chaque discontinuité de la visibilité correspond alors à une configuration le long de l'arête qui départage deux facettes élémentaires. Dans ce cas, il est donc nécessaire d'effectuer

$n_{change}+1$ tests de facettes, où n_{change} correspond au nombre de changements d'orientation constatés.

Remarquons que pour le cas des systèmes à ddls circulaires, le triplet de configurations considéré forme une facette de nature différente d'un point de vue topologique car celle-ci ne contient pas d'intérieur (c.f figure 4.14(d)). Une conséquence directe de cette propriété est que si l'on permute les configurations q_v, n_1 et n_2, les facettes élémentaires testées ne couvrent plus la même portion de l'espace (contrairement au cas des systèmes ne comportant pas de degré de liberté circulaire). Il est donc nécessaire de distinguer q_v de n_1 et n_2, pour déterminer les facettes élémentaires à tester.

Test élémentaire de facette : Pour savoir si une facette élémentaire appartient à $\mathcal{C}_{free}$, on utilise un algorithme dichotomique qui tente de recouvrir la facette de boules libres de l'espace des configurations (c.f. figure 4.15). La méthode de calcul de boules libres de $\mathcal{C}$ se fait en considérant la cinématique du robot et sa distance aux obstacles dans l'espace de travail. Cette méthode sera détaillée annexe B. L'algorithme de recouvrement débute en calculant le rayon de chacune des boules centrées sur les sommets respectifs de la facette. Si ces rayons permettent de recouvrir la facette, celle-ci appartient à $\mathcal{C}_{free}$. Sinon, on décompose la facette en deux sous facettes, de telle sorte que le nouveau sommet commun aux deux sous facettes soit le plus éloigné possible des zones déjà couvertes par les boules. Le rayon de la boule centrée sur le nouveau sommet est à son tour calculé et la décomposition se poursuit, jusqu'à ce qu'un sommet soit testé invalide ou que toute la facette soit recouverte.

4.3 Test de rétraction

La section précédente présentait un moyen de calculer la connexité d'un sous réseau visible à partir d'une configuration q_v. Lorsque le sous réseau n'est pas connexe, avant d'insérer q_v dans le réseau, on teste si ce nouveau nœud et les arêtes auxquels il sera connecté ne va pas participer à la création d'un chemin redondant. On construit donc le chemin $\tau = n_1 - q_v - n_2$, où n_1 et n_2 appartiennent à deux composantes distinctes du sous réseau, puis on teste la rétraction de ce chemin sur le réseau au moyen de l'algorithme *TestRetract* (ligne 9 de l'algorithme de Retract_PRM). L'algorithme *TestRetract* est représenté figure 4.16. Au cours de la recherche, l'algorithme extrait puis teste les différents chemins successifs de G, en commençant par les plus courts. Ce processus de recherche s'arrête dès qu'une rétraction a été trouvée ou lorsque tous les chemins possibles on été testés. Pour des raisons d'efficacité, on se limite en pratique aux N chemins les plus courts dans le réseau reliant n_1 à n_2.

La fonction *IsRetract* (ligne 3 de l'algorithme 4.16) permet de déterminer si deux chemins τ, τ' sont mutuellement rétractables. Cette fonction est basée sur le calcul du diagramme de visibilité entre les deux chemins par une méthode de grille. La rétraction est possible lorsqu'il existe un chemin entre les points $(0,0)$ et $(1,1)$ du diagramme (c.f. section 2.2). A partir d'un pas de discrétisation donné, on peut tester pour chaque couple $(i,j) \in [\![0,n]\!]^2$ la visibilité du chemin reliant $q_{t_i} = \tau(\frac{i}{n})$ à $q_{t'_j} = \tau(\frac{j}{n})$ et construire ainsi le diagramme de visibilité en testant

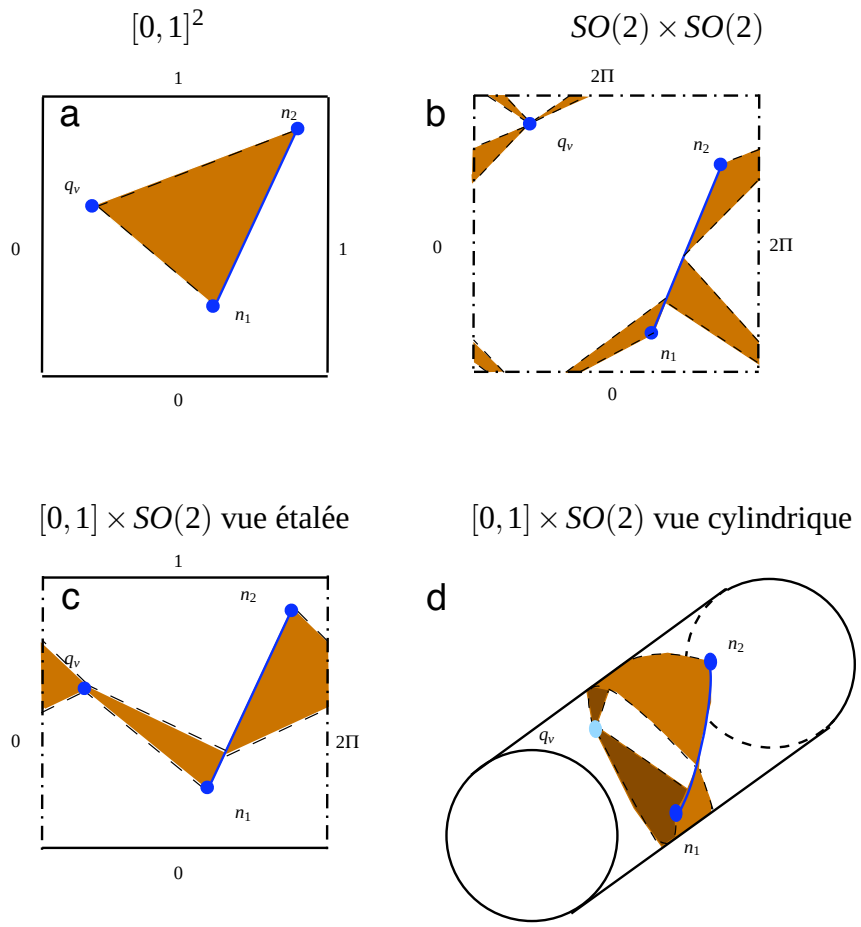


FIG. 4.14 – *Le test de visibilité d'une arête peut se décomposer en plusieurs tests élémentaires de facettes en fonction des ddls circulaires. Pour un espace de type* $[0,1]^2$ *(en haut à gauche), un seul test suffit. Pour l'espace* $SO(2) \times SO(2)$ *(en haut à droite), il est nécessaire de décomposer la surface en trois facettes (si on recolle les morceaux). Les figures du bas représentent une décomposition en deux facettes, pour un espace de type* $[0,1] \times SO(2)$*. La première figure est une vue étalée de la vue cylindrique de droite.*

la validité du chemin $\mathcal{L}(q_{t_i}, q_{t'_j})$. En pratique, on ne calcule pas l'intégralité de ce diagramme. Les tests sont limités aux éléments de la grille visités lors de la recherche d'un chemin "sans collision". La figure 4.17 montre que ce calcul incrémental du diagramme de visibilité permet de n'appliquer le test que sur un nombre très limité de pixels du diagramme et donc de rendre la méthode bien plus performante.

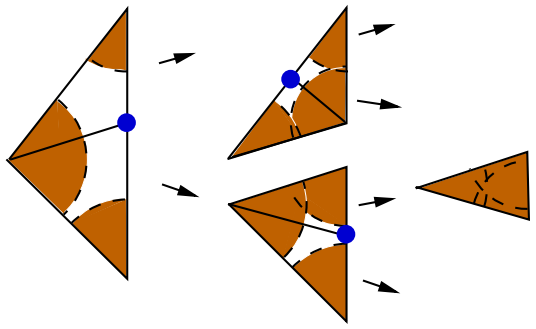


FIG. 4.15 – *Recouvrement dichotomique d'une facette de l'espace des configurations à l'aide de boules libres.*

TestRetract(τ, n_1, n_2, G)
1 $\tau' \leftarrow$ BestPath(n_1, n_2, G)
2 **While** $\tau' \neq \varnothing$
3 **If** IsRetract(τ, τ') $= 1$
4 Return 1
5 **End If**
6 $\tau' \leftarrow$ BestPath(n_1, n_2, G)
7 **End While**
8 Return 0

FIG. 4.16 – *Algorithme testant la rétraction sur le réseau d'un chemin formé à partir de q_v.*

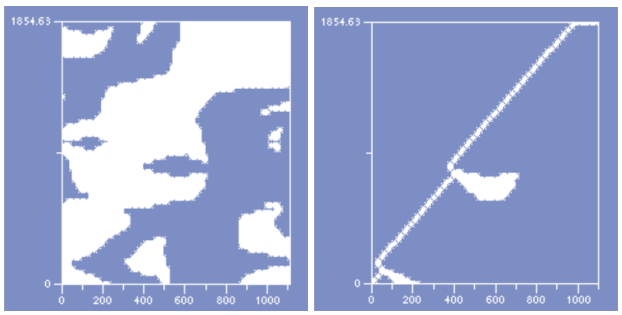


FIG. 4.17 – *Diagramme de visibilité (gauche) et portion réellement calculée (droite) pour un test de rétraction entre deux chemins.*

5 Résultats

Comme pour le planificateur à environnement changeant, la méthode de création des réseaux de rétraction a été implémentée sur la plate-forme Move3D. Les simulations ont ici été réalisées

à partir d'un PowerPC G4 à 1.2GHz sous Mac OS-X. Les tests de simulation présentés dans cette section montrent la généricité de la méthode de construction de réseaux de rétraction. Les résultats sont présentés en distinguant, trois types d'espaces des configurations différents : espaces bidimensionnels (x,y), espaces tridimensionnels (x,y,θ) et espaces tridimensionnels (x,y,z). Pour chaque type d'espace, plusieurs environnements sont présentés qui correspondent à des classes de problèmes différents. Dans chaque cas, les figures montrent l'arbre initial G_{vis} construit avec la méthode Visib-PRM et le réseau de rétraction final $G_{retract}$ construit avec l'algorithme Retract_PRM. En fin de section, un tableau récapitulatif (figure 4.21) présente les données associées à la construction des deux types de graphes. Notons que la méthode de création des réseaux de rétraction fait que chaque nouveau cycle correspond à l'ajout d'un noeud et de deux arêtes dans la structure de données.

Environnements 2D translation (figure 4.18) : Dans ces exemples, le robot est un cylindre en translation dans le plan. La première scène (A), représente un cas d'environnement simple. Le réseau de rétraction capture les classes d'homotopie mais ne rajoute pas de cycle supplémentaire par rapport à ces classes. En effet, en 2D, les opérations de rétraction couvrent une grande partie des opérations de déformation homotopique envisageables (i.e : beaucoup de chemins homotopes sont facilement déformables entre eux).

La scène suivante (B) est un environnement de type labyrinthe. Plus de 20 cycles sont nécessaires pour obtenir les classes d'homotopie. L'algorithme de rétraction permet de construire un réseau capturant l'ensemble de ces cycles en seulement 109 secondes. Pour la même raison que l'exemple précédent, la méthode rajoute très peu de cycles supplémentaires par rapport aux classes d'homotopie.

La dernière scène (C) possède une complexité géométrique importante : les nombreux objets présents dans la scène nécessitent un total de plus de 70 000 facettes. On voit que même pour des géométries complexes, l'algorithme calcule le réseau de rétraction avec un temps raisonnable seulement 4 fois plus grand que le temps de calcul d'un arbre de Visibilité (164s contre 41s).

Environnements 2D translation-rotation (figure 4.19) : Ces exemples concernent des robots à trois degrés de liberté : deux translations et une rotation. La scène (D) est la même que la scène (1), mais pour une barre (x,y,θ). Contrairement aux exemples 2D translation, le réseau de rétraction est plus riche que les classes d'homotopie. Ceci traduit la plus grande complexité de l'espace des configurations et donc la plus grande variété de chemins de l'espace que le réseau capture grâce à la méthode de rétraction.

La scène suivante (E) représente un problème de franchissement de passage étroit pour un carré. Quatre types de franchissement sont possibles, pour quatre orientations différentes du carré (chacune d'elles obtenue par rotation d'angle $\frac{\pi}{2}$). Ces quatre types de franchissement définissent quatre classes d'homotopie différentes dans l'espace des configurations. La capture de ces classes d'homotopie est un problème difficile et à notre connaissance, il n'existe aucune méthode probabiliste capable de faire cette capture en un temps raisonnable. L'algorithme de

construction de réseaux de rétraction permet de capturer ces classes de chemins en 37 secondes seulement. Les réseaux de visibilité et de rétraction sont présentés figures (E-a) et (E-b). La figure (E-c), est une vue agrandie du réseau de rétraction au niveau du passage étroit. La figure (E-d) représente le réseau dans un espace (x,y,θ) ce qui permet de distinguer les 4 types de franchissement du passage étroit associés aux 4 classes d'homotopie.

Il est intéressant de comparer ce résultat à celui obtenu à partir d'une méthode traditionnelle. Le tableau ci-dessous présente pour le même environnement (E), le nombre de classes d'homotopie capturées (sur un total de 4) et le temps de calcul pour la méthode de connexion aux k plus proches voisins et pour différents couples (N,k) choisis (N, nombre total de nœuds). Les données correspondent à une moyenne réalisée sur 5 essais. On constate que même pour le réseau du tableau le plus dense et redondant ($N = 8000$, $k = 100$), les classes d'homotopie ne sont pas systématiquement capturées (n_classes$= 3.2/4$), alors que le temps de calcul (t$= 3819$s) est plus de 100 fois supérieur à la méthode de rétraction qui capture les 4 classes en 37 secondes seulement. Ce résultat montre la nette efficacité de la méthode de rétraction, comparativement aux méthodes traditionnelles pour capturer les chemins peu semblables de l'espace et en particulier les chemins non homotopes.

		n_classes			temps (s)		
		k			k		
		10	20	100	10	20	100
N	500	0.2	0.6	0.4	2.1	3.5	15.3
	1000	0.	0.2	1.2	6.4	9.3	33.2
	2000	0.	0.6	1.6	33.2	43.5	110.0
	4000	0.8	1.0	2.8	246	336	455
	8000	1.4	2.4	3.2	2947	3295	3819

Environnements 3D (figure 4.20) : La première scène (F) permet de montrer un réseau de rétraction pour une géométrie 3D simple composée d'obstacles sphériques. Le réseau de rétraction contient 9 cycles élémentaires alors que l'espace ne contient pour cet exemple qu'une classe d'homotopie. Ce résultat montre que la méthode de rétraction permet de construire des réseaux de faible densité tout en capturant une variété de chemins nettement plus grande que les classes d'homotopie. Les exemples (G) et (H) représentent des scènes 3D de complexité croissante. Ils mettent en évidence que même pour des problèmes plus complexes, les réseaux de rétraction permettent à l'aide d'une structure à la fois riche et compacte de capturer les principales variétés de chemins de l'espace. Le temps associé à la construction de ces réseaux reste très limité, de l'ordre d'une minute de calcul.

Ces tests montrent que les réseaux de rétraction sont un outil puissant, permettant à travers une structure de données réduite et avec de très bonnes performances, de capturer les différents types de chemins de l'espace difficilement déformables entre eux.

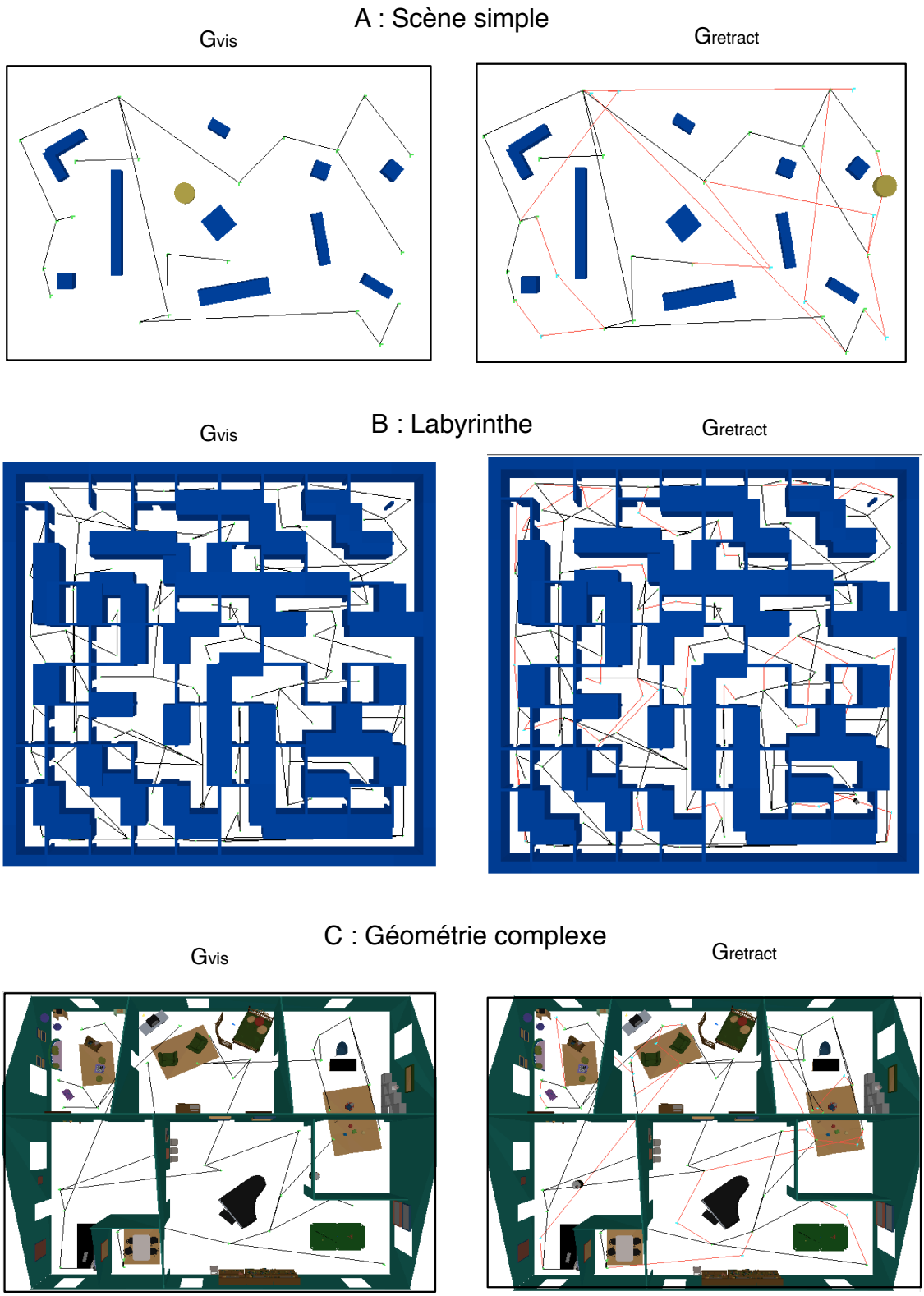


FIG. 4.18 – *Environnements 2D translation.*

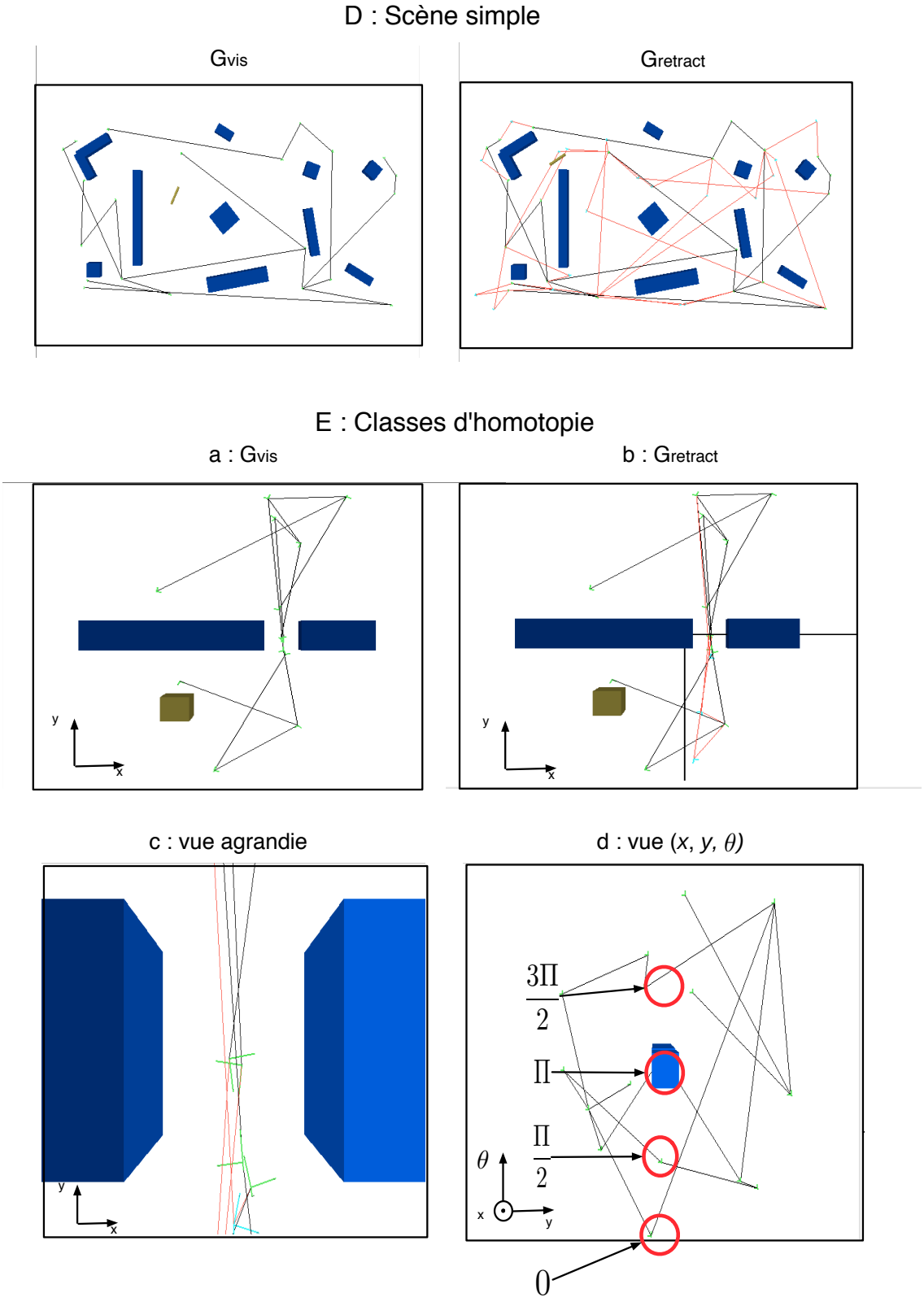


FIG. 4.19 – *Environnements 2D translation-rotation.*

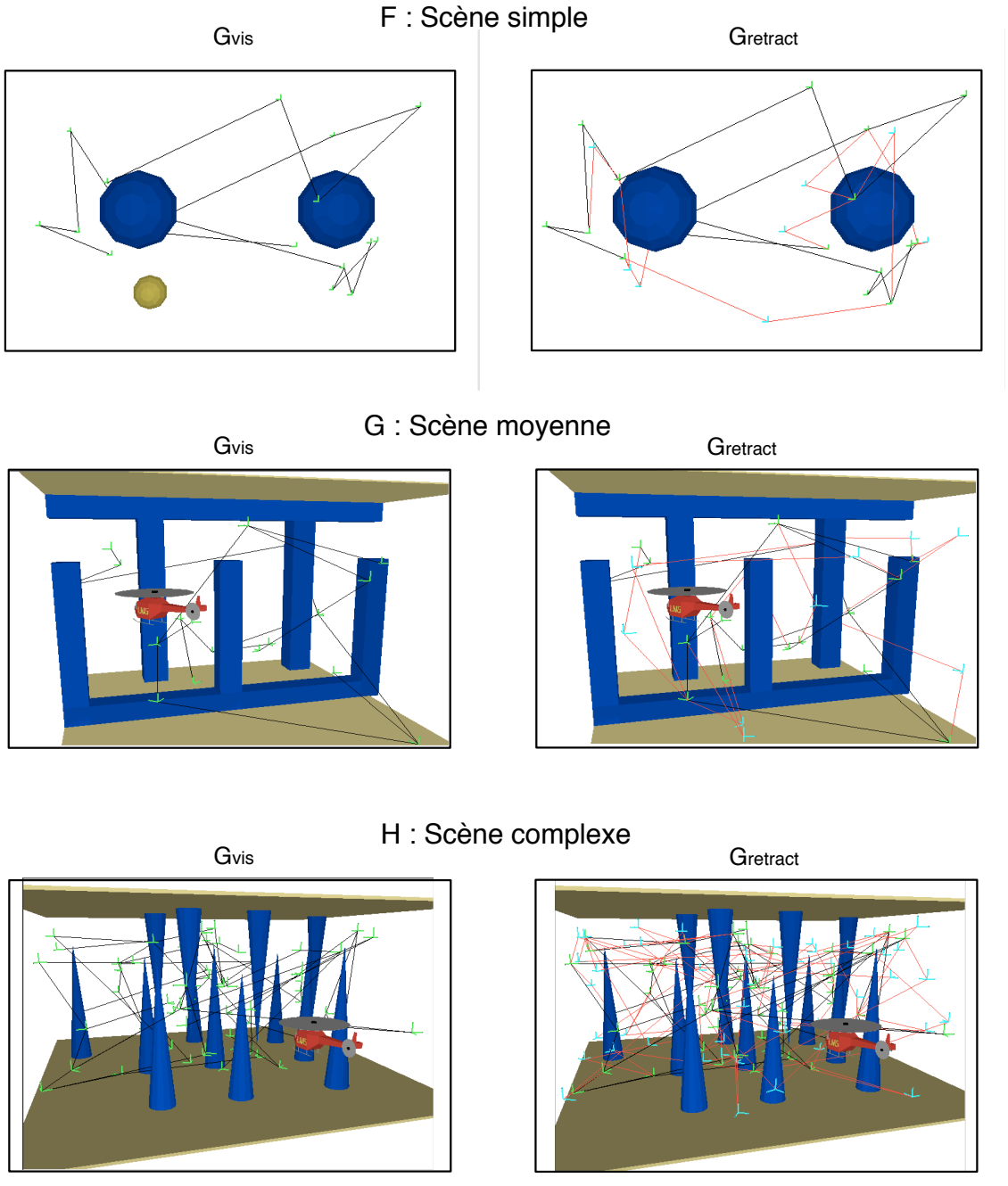


FIG. 4.20 – *Environnements 3D.*

		Environnements							
		A	B	C	D	E	F	G	H
G_{vis}	temps (s)	0.7	21	41	1	9	1	3	13
	N	21	120	48	23	9	17	20	55
	E	20	119	47	22	8	16	19	54
$G_{retract}$	temps (s)	13	109	164	26	37	22	56	175
	N	31	149	66	42	12	28	30	113
	E	40	177	83	60	14	34	39	170
	n_cycles	10	29	18	19	3	9	10	58

FIG. 4.21 – *Tableau récapitulatif des résultats expérimentaux.*

Chapitre 5

Réseaux Robustes aux Changements de Contexte

Les réseaux présentés au chapitre précédent permettent de capturer à l'aide d'une structure de données compacte, les classes de chemins potentiellement intéressants. Vis à vis du planificateur à environnements changeants, la capture de ces différentes classes de chemins, combinée à des opérations de reconnexions locales performantes, offre la possibilité de calculer rapidement des solutions alternatives dans le réseau en présence d'objets dynamiques.

Dans ce chapitre nous décrivons une méthode qui intègre la prise en compte des obstacles dynamiques dans le calcul du réseau. Ici, on considère donc que les caractéristiques relatives aux obstacles mobiles font partie des données d'entrée du problème. La méthode proposée intègre cette information lors de la construction du réseau, permettant d'obtenir cette fois-ci des garanties quant à la robustesse vis à vis des obstacles mobiles. On parlera alors de *réseau robuste aux changements de contexte*. Le formalisme et les résultats développés prennent pour support les travaux initialement proposés dans [van den Berg 05a], fruit d'une réflexion menée en collaboration avec l'université d'Utrecht. La contribution de ce chapitre est de proposer une formalisation plus développée de ces problèmes. La "mise en œuvre expérimentale" a été réalisée par l'équipe d'Utrecht et n'est donc pas le fruit direct de notre travail.

Ce chapitre se compose de trois sections. La première décrit de façon formelle la problématique liée aux réseaux robustes aux changements de contexte. La seconde propose une méthode algorithmique de construction de ce type de réseau. Enfin, la dernière propose une validation expérimentale à travers différents exemples de scénarios.

1 Formalisme

1.1 Hypothèse sur les placements

On considère que chaque obstacle mobile est un corps rigide de la scène 3D. On peut alors lui associer un vecteur de $[0,1]^3 \times SO(3)$, définissant sa position et son orientation dans l'espace. Dans de nombreux cas, l'ensemble des configurations envisageables pour l'obstacle peut cependant ne correspondre qu'à un sous-ensemble de $[0,1]^3 \times SO(3)$ et peut donc être modélisé par un espace des configurations de dimension inférieure. Considérons par exemple les environnements représentés figure 5.1. Le premier exemple correspond à un obstacle mobile "porte", pour lequel une représentation à deux états (ouvert/fermé) s'avère suffisante. Le sous-ensemble des configurations envisageables ne comporte alors que deux éléments. Le deuxième exemple correspond à un exemple de type "ligne de bus", pour lequel les positions possibles sont situées sur un chemin τ de $[0,1]^3 \times SO(3)$. Dans ce cas, les différentes configurations de l'obstacle mobile se ramènent à un espace de dimension 1 (abscisse curviligne le long d'une trajectoire). Finalement, le dernier environnement illustre le cas d'un objet dont le mouvement se limite à un mouvement plan. On peut alors lui associer un espace des configurations de dimension 3 de type (x,y,θ), où les bornes des paramètres x et y correspondent aux limites de la pièce.

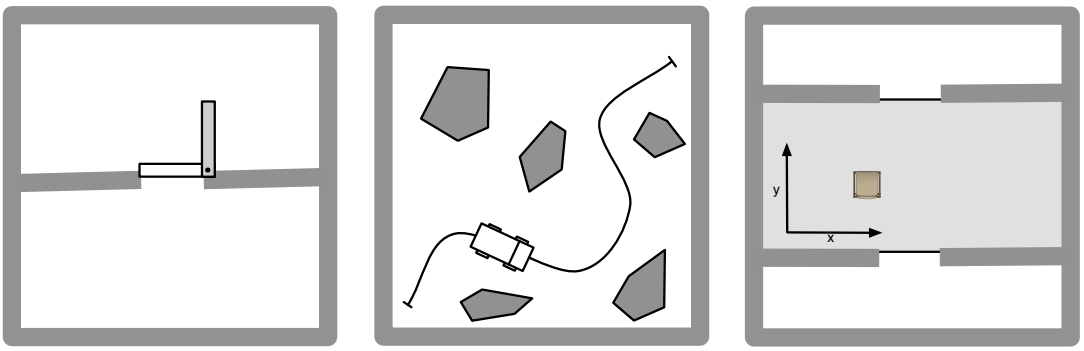


FIG. 5.1 – *Exemples pour lesquels les positions possibles ne représentent qu'une partie de l'ensemble des positions envisageables au sein de l'environnement.*

On supposera par la suite que chaque obstacle mobile O vérifie ce type de restriction et qu'il est ainsi possible d'obtenir une représentation $P(O)$, des placements possibles de l'obstacle. On notera $p_o \in P(O)$ un placement particulier.

1.2 Notion de contexte

Considérons un environnement composé de K obstacles mobiles $O_0, \ldots, O_{K-1}$. Alors l'ensemble $p = \{p_0, , \ldots, p_{K-1}\}$ avec $p \in P = (P(O_0) \times \ldots \times P(O_{K-1}))$
sera appelé un *contexte* particulier de scène.

1.3 Connexité invariable et plages de connexion

Deux configurations q_i et q_j de $\mathcal{C}$ sont dites *invariablement connexes*, si elles vérifient les propriétés suivantes :

1. Les validités de q_i et de q_j sont les mêmes pour tout contexte.
2. Pour tout contexte p tel que q_i et q_j valides, alors q_i et q_j connexes.

Si deux configurations ne sont pas invariablement connexes, on peut leur associer une *plage de connexion* $F(q_i, q_j)$ représentant l'ensemble des contextes pour lesquels elles sont connexes :

$$F(q_i, q_j) = \{p \in P |\ q_i, q_j \text{ connexes}\}$$

Par extension, on regroupe l'espace libre privé des obstacles mobiles en différentes *composantes invariablement connexes* notées $Comp_i$, (éventuellement une infinité), constituées des ensembles de configurations invariablement connexes. De même on définit une plage de connexion entre deux composantes invariablement connexes distinctes comme étant la plage de connexion des configurations qui la composent :

$$F(Comp_i, Comp_j) = F(q_i, q_j) \text{ avec } q_i \in Comp_i \text{ et } q_j \in Comp_j$$

De manière similaire, on définit ces différentes relations de connexité au niveau d'un réseau. Ainsi, deux nœuds sont dits invariablement connexes si, lorsqu'ils sont valides, les deux nœuds sont connexes **au sein du réseau** quel que soit le contexte. On définit également une plage de connexité entre nœuds non invariablement connexes :

$$F(n_i, n_j) = \{p \in P |\ IsConnect(n_i, n_j, p) = 1\}$$

où la fonction $IsConnect(n_i, n_j, p)$ vaut 1 quand il existe pour le contexte p un chemin valide du réseau reliant n_i à n_j. Finalement les nœuds du réseau sont regroupés en différentes composantes invariablement connexes notées L_i et on définit des plages de connexion entre ces composantes :

$$F(L_i, L_j) = F(n_i, n_j) \text{ avec } n_i \in L_i \text{ et } n_j \in L_j$$

Considérons l'exemple de la figure 5.2. L'obstacle mobile est un carré situé soit en a, soit en b. Le réseau pour un robot ponctuel comporte alors 3 composantes invariablement connexes : L_1, ensemble des nœuds invalides quand l'obstacle est en a, valides et connexes sinon. L_2,

ensemble des nœuds invalides quand l'obstacle est en *b*, valides et connexes sinon. Enfin L_3, ensemble des nœuds valides et connexes quel que soit le contexte.

Cet exemple permet également de faire ressortir une propriété propre aux composantes invariablement connexes : contrairement aux composantes connexes classiques des environnements statiques, deux nœuds appartenant à la même composante invariablement connexe peuvent être reliés par un chemin composé de nœuds n'appartenant pas à cette composante. Sur la figure, c'est le cas pour la composante L_3 : les nœuds de la partie supérieure et ceux de la partie inférieure sont reliés en passant par des nœuds appartenant soit à L_1, soit à L_2.

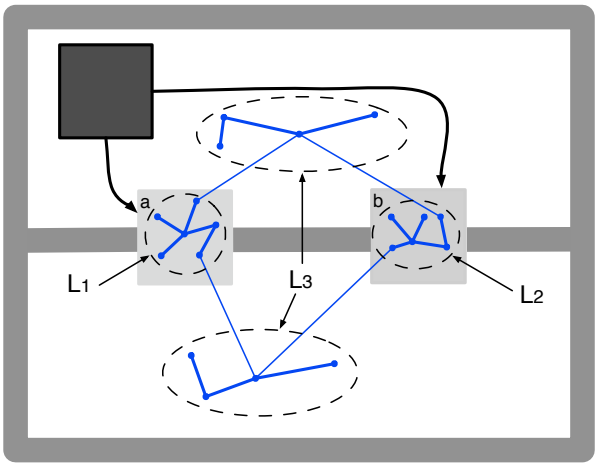


FIG. 5.2 – *Réseau formé de trois composantes invariablement connexes (L_1,L_2,L_3) pour un obstacle mobile pouvant prendre deux positions a et b.*

1.4 Réseau robuste aux changements de contexte

On dira d'un réseau qu'il est *robuste aux changements de contexte* (en probabilité), si quand le temps t tend vers l'infini, sa connexité courante correspond à la connexité de l'espace libre, quel que soit le contexte. Autrement dit, le réseau tend à être complet en probabilité pour tout type de contexte.

En reprenant les définitions précédemment introduites, cela signifie que les plages de connexion des composantes invariablement connexes du réseau tendent, quand t tend vers l'infini, vers les plages de connexion des composantes invariablement connexes de $\mathcal{C}$.

Nous présenterons au cours de la section suivante une méthode permettant de construire à l'aide d'un nombre réduit d'arêtes et de nœuds de tels réseaux.

2 Construction du réseau

2.1 Regroupement en parcelles

Afin de rendre possible un traitement algorithmique du problème, nous proposons de faire une partition des sous-placements possibles que possède un obstacle mobile. On appellera les différentes portions découpées des *parcelles*. Ainsi chaque obstacle mobile O_i, sera découpé en n_i parcelles $\chi_i^0 \dots \chi_i^{n_i-1}$ avec :

$$\bigcup_j \chi_i^j(O_i) = P(O_i) \quad \text{et} \quad \chi_i^j \cap \chi_i^k = \emptyset \text{ si } j \neq k.$$

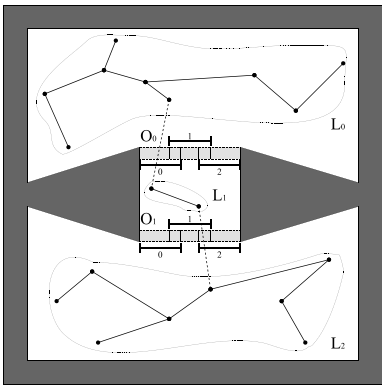
Ce découpage doit cependant posséder certaines caractéristiques minimales pour assurer la construction de réseaux robustes. En particulier, celui-ci devra respecter les propriétés suivantes :

1. Toutes les positions associées à une parcelle donnée doivent correspondre (du point de vue du robot) à un espace libre de même connexité.
2. La connexité de l'espace libre en prenant pour obstacle la parcelle dans son intégralité, doit être égale à celle de l'espace libre pour chacune des positions associées à cette parcelle.

En pratique ce découpage s'avère facile à réaliser dans deux types de situations : d'abord, quand l'obstacle mobile possède un nombre fini de positions. Il suffit alors de prendre pour parcelles chacune des positions discrètes de l'obstacle. Ensuite, quand la connexité de l'espace libre ne peut être cassée par l'obstacle mobile. Dans ce cas, il suffit de prendre des parcelles suffisamment petites pour qu'elles-mêmes, ne puissent casser la connexité de $\mathcal{C}_{free}$. Dans des situations plus complexes, ce découpage peut nécessiter une plus grande attention.

Le regroupement des positions en parcelles permet d'obtenir une représentation condensée et discrétisée de l'ensemble des placements d'un obstacle mobile. A partir de ces parcelles, il est possible de construire une représentation des plages de connexion entre composantes : on construit un tableau K-dimensionnel booléen (K, nombre total d'obstacles mobiles) où chaque cellule de ce tableau correspond à l'état de connexité des deux composantes pour un contexte particulier. La figure 5.3 montre un exemple de plages de connexion pour un problème à deux obstacles (le tableau est donc bidimensionnel). Pour chaque obstacle, on découpe l'ensemble des placements possibles en 3 parcelles. La scène comporte donc $3^2 = 9$ contextes différents. Chaque cellule du tableau représente alors la validité du réseau pour un contexte donné.

Quand la plage de connexion entre deux composantes est telle que le tableau booléen associé est entièrement rempli de 1, on dit que la plage est complète. Deux composantes dont la plage de connexion est complète sont en fait connexes quel que soit le contexte et représentent donc une seule et même composante.

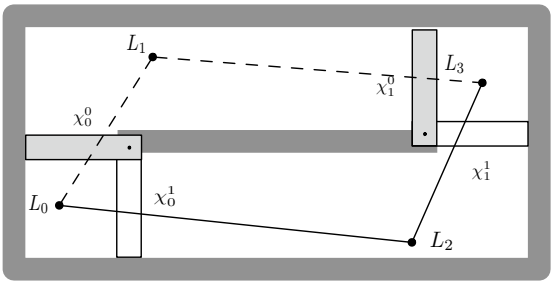


$F(L_0, L_2)$

	χ_0^0	χ_0^1	χ_0^2
χ_1^0	0	0	0
χ_1^1	0	0	0
χ_1^2	0	1	1

FIG. 5.3 – *Dans cet exemple, les placements des obstacles mobiles (rectangles en translation suivant les régions gris clair), sont regroupés en 3 parcelles de taille identique. Ce découpage permet d'obtenir une représentation binaire des plages de connexion $F(L_0, L_2)$ entre la composante haute (L_0) et basse (L_2) de l'environnement.*

Une autre remarque importante est que les plages de connexité entre composantes ne peuvent pas être déduites des plages de connexité calculées en considérant les obstacles mobiles indépendamment. Il y a en effet interdépendance entre les obstacles vis à vis de la connexité globale de l'espace libre. Ce phénomène est illustré à l'aide de l'environnement de la figure 5.4. Chaque porte peut prendre deux types de positions possibles. On s'aperçoit que la connexité entre les composantes L_0 et L_3 dépend de la combinaison des deux positions de porte et ne peut être exprimée en considérant chacune d'elles de manière indépendante.



$F(L_0, L_3)$

	χ_0^0	χ_0^1
χ_1^0	1	0
χ_1^1	0	1

FIG. 5.4 – *Exemple élémentaire illustrant l'interdépendance des plages de connexion vis à vis des obstacles mobiles.*

2.2 Algorithme

L'algorithme global de construction d'un réseau robuste aux changements de contexte est présenté figure 5.5. A chaque itération, une configuration q est échantillonnée aléatoirement et s'il existe au moins un contexte pour lequel q est valide, elle est rajoutée au réseau en tant que nœud (ligne 7). A ce stade de l'algorithme le nœud formé représente à lui tout seul une composante invariablement connexe L_q dont les plages de connexion avec les autres composantes sont initialisés au moyen de tableaux K dimensionnels remplis de 0 (lignes 8 à 12). Ensuite, l'algorithme teste pour chacun des nœuds présents dans le réseau, s'il est nécessaire d'effectuer une connexion avec le nœud nouvellement inséré. Ces tests sont réalisés au moyen de la fonction *Necessary* (ligne 14). Lorsqu'une connexion est nécessaire, une arête est rajoutée et les plages de connexité des différentes composantes du réseau sont mises à jour de façon itérative à l'aide d'un algorithme de propagation. Cet algorithme correspond à la fonction *Propagate* utilisé ligne 17. Si à la suite de cette mise à jour, la plage de connexion entre deux composantes devient complète (i.e. le tableau K-dimensionnel associé est entièrement rempli de 1), alors elles sont fusionnées.

Au cours des trois sous-sections suivantes, nous présenterons la méthode de test des arêtes utiles (fonction *Necessary*), la méthode de propagation des plages de connexité (fonction *Propagate*), et enfin la méthode de test de validité d'un nœud ou d'une arête vis à vis d'une parcelle.

Critère d'ajout d'une arête

Après avoir inséré un nouveau nœud n_q, la fonction *Necessary* teste pour tout nœud n_i déjà présent dans le réseau, si l'arête $e_{n_q n_i}$ est nécessaire. Les arêtes dites *nécessaires* sont celles qui augmentent la connexité du réseau. Une arête est donc rajoutée au réseau seulement si elle connecte deux composantes invariablement connexes différentes et si elle permet d'étendre la plage de connexion associée à ces deux composantes. Il faut donc que l'on ait :

$$\{p \in P | TestEdge(e,p) = 1\} \not\subset F(L_0, L_1)$$

Pour savoir si l'arête vérifie cette propriété, on teste d'abord si elle est valide par rapport à l'environnement statique. Si elle l'est, on examine sa validité par rapport aux obstacles mobiles pour les différents contextes pour lesquels les deux composantes ne sont pas encore reliées (représenté par des 0 dans le tableau K-dimensionnel). Si pour un des contextes, l'arête est valide, cela signifie qu'elle augmente la plage de connexion des composantes. Elle est donc insérée dans le réseau.

Propagation des plages

Après l'ajout d'une arête reliant deux composantes, il est nécessaire de mettre à jour la plage de connexion relative aux deux composantes, puis de propager l'information de connexion aux plages reliant d'autres paires de composantes connexes. Ce processus est réalisé par l'algorithme *Propagate* présenté figure 5.6. Celui-ci prend en entrée les deux composantes pour lesquelles la connexité est mise à jour et les contextes $P_f \subset P(O)$ pour lesquels l'arête ajoutée

CONSTRUCT_ROBUST_ROADMAP

input : the robot A, the environment B, the mobile obstacles O_i with the set of context P, the maximal number of nodes n_{max}

output : the roadmap G, the set of valid connections between components

```
1   G ← EmptyRoadmap
2   n ← 0
3   ListComp ← ∅
3   While n < n_max
4       q ← RandomConfig(A, B)
5       If {∃p ∈ P | ColFree(q, p)}
6           n ← n + 1
7           AddNode(q, G)
8           L_q ← NewComp(q)
9           AddNewComp(ListComp, L_q)
10          For All Composante L_i ≠ L_q
11              F(L_i, L_q) ← InitSet
12          End For All
13          For All n_i ∈ G, n_i ≠ n_q
14              If Necessary(e_{n_i n_q})
15                  AddEdge(e_{n_i n_q}, G)
16                  P_f ← {p ∈ P|ColFree(e_{n_i,n_q}, p)}
17                  Propagate(L_q, L_{n_i}, P_f)
18                  MergeComps(G)
19              End If
20          End For All
21      End If
22  End While
```

FIG. 5.5 – *Algorithme de construction d'un réseau robuste.*

est libre. L'information de connexion se propage de composantes voisines en composantes voisines à partir des deux composantes initiales auxquelles est reliée l'arête (deux composantes sont voisines si elles possèdent une arête qui les relie).

Tests de validité et parcelles

Lors du processus de construction du réseau, différentes plages de validité de configurations (ligne 5 figure 5.5) et d'arêtes (fonction *Necessary*) sont calculées. Celles-ci s'appuient sur des tests de collision vis à vis des différentes parcelles prises pour découpage. Pour réaliser ces tests, on construit pour chaque parcelle un objet couvrant l'ensemble du volume balayé par l'obstacle mobile quand celui-ci parcourt les différentes positions inclues dans la parcelle. Pour y parvenir, on discrétise les positions de l'obstacle mobile à l'intérieur de la parcelle et on utilise un recouvrement de la scène à l'aide d'une grille de voxels. L'objet représentatif d'une parcelle est alors formé en regroupant les différents voxels balayés par l'obstacle mobile pour les différents placements discrets testés.

```
PROPAGATE (L_a, L_b, P_f)
1   F(L_a, L_b) ← F(L_a, L_b) ∪ P_f
2   For All neighbors L_i of L_a
3       P_f ← F(L_a, L_b) ∩ F(L_a, L_i)
4       If P_f ⊄ F(L_b, L_i)
5           PROPAGATE (L_b, L_i, P_f)
6       End If
7   End For All
8   For All neighbors L_i of L_b
9       P_f ← F(L_a, L_b) ∩ F(L_b, L_i)
10      If P_f ⊄ F(L_a, L_i)
11          PROPAGATE (L_a, L_i, P_f)
12      End If
13  End For All
```

FIG. 5.6 – *Propagation des plages de connexion entre paires de composantes invariablement connexes*

2.3 Complétude probabiliste

Les réseaux construits au moyen d'un tel algorithme tendent à être robustes aux changements de contexte, c'est à dire qu'ils sont complets en probabilité pour tout type de contexte.

La démonstration de cette propriété s'appuie sur la démonstration standard de complétude probabiliste des méthodes PRM [Svestka 97a]. Considérons un environnement composé d'obstacles statiques et mobiles et une requête $(init, goal, p)$ pour laquelle il existe un chemin solution. Nous pouvons couvrir ce chemin de boules se chevauchant et appartenant à $\mathcal{C}_{free}$. Soient q_i, q_j, appartenant à deux intersections successives de boules. La connexion entre ces deux configurations est libre. Il existe alors deux cas : soit cette connexion est *nécessaire* et l'arête reliant les deux configurations sera rajoutée au réseau, soit elle ne l'est pas et il existe déjà pour le contexte considéré un chemin sans collision qui relie les deux configurations. Dans les deux cas, les deux configurations sont au final connectées. Comme la probabilité qu'une configuration soit tirée dans chacune des intersections de boules tend vers 1 quand le temps tend vers l'infini, on est assuré de trouver au final un chemin entre *init* et *goal* pour tout contexte p donné.

2.4 Mémorisation des plages

Au cours de la construction d'un réseau robuste, on teste pour différents contextes l'état de validité des arêtes insérées (ligne 16 de l'algorithme 5.5). Cette information qui est mémorisée lors de la construction des réseaux pour maintenir l'information de connexité entre composantes peut également être exploitée lors des phases de requêtes. Au cours du chapitre 2 section 3.5, nous avons déjà présenté une structure spécifique permettant de stocker des informations de collision au niveau des arêtes. A l'aide de cette structure d'arête, l'information de validité

pourrait être stockée dès la phase de construction du réseau initial et permettre ainsi de réduire le coût des tests de collisions, dès les premières requêtes soumises au planificateur.

3 Résultats

L'algorithme de construction de réseaux robustes a été implémenté en C++, en utilisant SOLID [van den Bergen 04] comme détecteur de collisions. Les expériences sont réalisées sur un Pentium 3.0GHz. Pour des raisons de performance, les tentatives de connexion entre nœuds ne sont réalisées que pour des distances inférieures à la moitié de la taille totale de la scène. Les temps de construction donnés correspondent à des temps moyens sur 25 tests. Les trois types d'environnements changeants expérimentés sont présentés ci-dessous.

Le premier environnement représenté figure 5.7 se compose de deux couloirs et d'une pièce centrale à l'intérieur de laquelle se situe une chaise. L'ensemble des placements possibles pour cet obstacle est décomposé en 8 parcelles. L'algorithme est stoppé quand les nœuds du couloir inférieur et ceux du couloir supérieur sont détectés comme faisant partie de la même composante. En effet, les couloirs inférieurs et supérieurs de l'espace forment ici une même composante invariablement connexe car ces portions de l'espace sont connectées quelles que soient les positions de la chaise. Pour cet exemple, le temps moyen de construction d'un réseau robuste est seulement de 0.25 secondes.

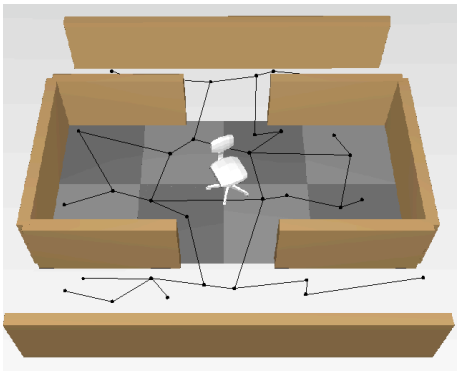


FIG. 5.7 – *Découpage en 8 parcelles pour un obstacle chaise et exemple de réseau construit à partir de ce découpage.*

La scène suivante (figure 5.8), comprend 4 portes, chacune pouvant prendre deux types de positions. Il existe donc au total $2^4 = 16$ types de contextes différents. Même si cela n'est pas facile à observer, les configurations *init* et *goal* représentées sur la figure peuvent être reliées quelles que soient les différentes combinaisons de placements des portes. L'algorithme de construction du réseau est stoppé quand ces deux configurations appartiennent à une même composante invariablement connexe. Là encore la méthode de construction du réseau robuste

est très performante : elle permet de construire un réseau au sein duquel *init* et *goal* sont invariablement connexes en seulement 0.19 secondes.

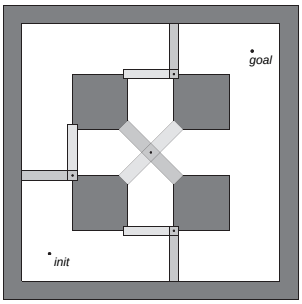


FIG. 5.8 – *Environnement 4 portes. La position goal est atteignable à partir de la position init quel que soit le contexte.*

Le dernier exemple est une variation de l'exemple précédent, mais la taille de la scène et le nombre d'obstacles mobiles est doublé (c.f figure 5.9). L'intérêt ici est d'estimer l'impact de l'augmentation de la complexité sur le temps de calcul, le nombre total de contextes étant de $2^8 = 256$. Comme précédemment, on arrête l'exploration quand les nœuds situés aux deux extrémités de l'environnement deviennent invariablement connexes. Malgré le nombre total de contextes à envisager, le temps de construction d'un tel réseau est seulement de 1.94 secondes. Notons également que le réseau de la figure 5.9 contient un nombre limité d'arêtes. Ceci est possible grâce au critère de sélection des arêtes à insérer, basé sur la notion d'arête nécessaire.

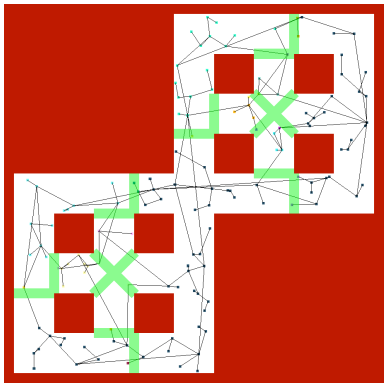


FIG. 5.9 – *Environnement 8 portes. La complexité provient du nombre total de contextes à envisager.*

Ces résultats permettent de mettre en évidence la rapidité de construction des réseaux robustes, même si les environnements présentés sont ici géométriquement très simples. Le dernier exemple montre en particulier qu'il est possible de traiter des environnements comportant de nombreux obstacles mobiles en conservant des temps de construction limités.

Nous avons présenté une méthode exploitant certaines connaissances sur les obstacles mobiles pour guider la construction des réseaux. Ces réseaux présentent au final la propriété forte d'être complets en probabilité pour tout contexte. Des résultats préliminaires montrent enfin la possibilité de construire de tels réseaux en des temps raisonnables, même lorsque le nombre d'obstacles mobiles est conséquent.

Conclusion et Perspectives

Les travaux présentés dans ce manuscrit portent sur la planification de mouvement pour des systèmes articulés dans des scènes partiellement dynamiques rencontrées dans de nombreuses applications, en particulier en robotique et en animation graphique. Les méthodes proposées conduisent à un planificateur réactif capable de prendre en compte efficacement les modifications de la scène.

Les méthodes développées sont génériques. Elles considèrent des systèmes articulés de complexité variable, dans des environnements géométriques 3D composés d'une partie statique et de divers types d'obstacles mobiles (placements discrets/continus, disparition/apparition ou changement de position). La généricité de notre approche vient de l'utilisation de méthodes probabilistes dont la performance a été démontrée dans de nombreuses applications. La contribution de cette thèse a porté sur leur extension pour la planification réactive dans des environnements partiellement dynamiques. On peut cependant noter que les méthodes algorithmiques proposées, en particulier les algorithmes DD-RRT et les réseaux de rétraction représentent une contribution qui dépasse le cadre des environnements changeants et constituent des outils performants directement exploitables dans le cas d'environnements statiques.

Une première contribution de la thèse est de proposer un planificateur dédié aux scènes dynamiques, intégrant la complémentarité des techniques probabilistes par réseau et par diffusion au sein d'une même architecture. Les réseaux de rétraction et les réseaux robustes aux changements de contexte sont deux méthodes permettant d'initialiser ce planificateur au moyen d'un réseau valide vis à vis de l'environnement statique. L'alternative entre ces deux méthodes dépend des données d'entrée du problème : lorsque les placements possibles des obstacles mobiles sont connus, il est possible de faire appel à la méthode des réseaux robustes pour construire de manière simple un réseau initial de taille réduite. Lorsque les obstacles mobiles (ou leur position) ne sont pas connus à l'avance, la méthode des réseaux de rétraction est capable d'assurer la construction d'un réseau initial à la fois riche et de taille réduite.

L'efficacité locale des techniques de diffusion est également exploitée pour reconnecter efficacement les arêtes invalidées par les obstacles mobiles. Enfin, la bonne performance de la mise à jour du réseau est assurée par des méthodes paresseuses combinées à un processus de mémorisation permettant de réduire considérablement le coût des tests de collision qui représentent les opérations les plus coûteuses.

L'introduction de domaines dynamiques dans l'algorithme de diffusion DD-RRT permet de contrôler le processus d'expansion en modifiant le domaine d'attraction des noeuds des arbres développés. La version "adaptative" permet de contrôler pour une meilleur robustesse, la taille des domaines dynamiques en exploitant l'information obtenue au cours du processus d'expansion. La performance de cet algorithme sur des problèmes localement contraints le rend tout à fait adapté aux opérations de reconnexions locales lors de la mise à jour dynamique du réseau.

L'introduction d'une notion de "rétraction", proche de l'homotopie, nous a permis de proposer une méthode originale de construction de réseaux, à la fois riches de par la grande diversité des chemins représentés et compactes grâce à la non redondance des cycles créés. Ces réseaux garantissent notamment de capturer les classes d'homotopie dans l'espace des configurations, problèmes pour lesquels les méthodes PRM s'avèrent peu performantes. Ces réseaux de rétraction présentent un intérêt aussi bien pour les environnements changeants que pour les scènes statiques. Dans le premier cas, ces réseaux offrent des solutions alternatives quand des chemins du réseau sont invalidés par la présence d'obstacles mobiles. Pour le second cas, les méthodes standards de lissage utilisées pour l'optimisation locale des chemins calculés dans le réseau, combinées aux différentes variétés de chemins de l'espace capturés par les réseaux de rétraction, offre la possibilité d'une optimisation plus globale des chemins solutions. De plus, contrairement à la plupart des méthodes PRM, les réseaux de rétractions intègrent, comme les réseaux de visibilité, un critère d'arrêt pertinent directement lié à la couverture de l'espace.

La méthode des réseaux robustes aux changements de contexte s'appuie sur une connaissance a priori des positions possibles des obstacles mobiles pour guider la construction de réseaux de taille réduite qui possèdent une propriété de complétude probabiliste, indépendamment des positions prises par les obstacles mobiles.

Les résultats décrits dans cette thèse ouvrent la voie à plusieurs améliorations et extensions.

Améliorations et Extensions

Exploiter la cohérence spatiale Le planificateur pour environnements changeants intègre un processus de mémorisation qui permet de tirer profit des tests de collisions effectués au cours des différentes requêtes. Lors d'une nouvelle requête, des tests de collisions sont évités si un des obstacles mobiles reprend un de ses anciens placements. Une amélioration de ce

mécanisme serait d'exploiter la cohérence spatiale des obstacles mobiles pour les nombreuses situations dans lesquelles la position de certains obstacles varie peu entre deux requêtes (en particulier, quand l'intervalle de temps séparant les deux requêtes est petit). En effet, dans ce cas la validité globale du réseau vis à vis de ces obstacles est peu altérée d'une requête à l'autre et seules les arêtes suffisamment proches de ces obstacles mobiles nécessitent d'être testées lors de la mise à jour du réseau.

Stratégies de reconnexion locale. La généricité des méthodes de diffusion et leur bonne performance locale ont été exploitées pour générer un outil de reconnexion des portions de réseau invalidées par les obstacles mobiles. Il est toutefois possible d'envisager des méthodes de reconnexion locales plus sophistiquées. Par exemple la configuration trouvée invalide lors de la mise à jour pourrait être exploitée pour guider l'opération de reconnexion. La limitation de la taille de la zone explorable pourrait également améliorer le processus de reconnexion, les méthodes de diffusion ayant tendance à s'étendre à des régions de l'espace trop importantes. Une autre alternative serait de disposer de "fuseaux" de chemins précalculés, permettant une reconnexion rapide des deux extrémités d'une arête invalidée. Les chemins de reconnexion utilisés pourraient également être obtenus par une mémorisation des tentatives de reconnexion ayant réussi.

Planification sous contrainte cinématique. Les résultats présentés dans ce manuscrit mettent en œuvre des systèmes mécaniques en l'absence de contrainte cinématique (e.g. méthode locale linéaire). Dans l'ensemble, les méthodes développées sont cependant généralisables à des systèmes soumis à des contraintes cinématiques, par exemple les systèmes non-holonomes. Ainsi, les méthodes de mise à jour et de mémorisation proposées au cours du chapitre 2, les réseaux robustes aux changements de contexte ainsi que les algorithmes DD-RRT sont utilisables pour des méthodes locales quelconques. Seule la méthode des réseaux de rétraction nécessite d'être étendue pour des systèmes cinématiquement contraints (la méthode de calcul des rétractions proposée repose sur une méthode locale linéaire). Une stratégie pour réaliser cette extension consiste à se baser sur des réseaux de rétraction construits à l'aide d'une méthode linéaire puis de les modifier pour les adapter à la méthode locale réellement mise en jeu. De même, l'adaptation des réseaux de rétraction à des systèmes à méthode locale orientée (i.e. chemins non réversibles) permettrait d'utiliser le planificateur dans des applications en animation graphique pour contrôler le mouvement de personnages virtuels dans des scènes dynamiques.

Qualité de chemin versus performance. Le planificateur permet d'obtenir des chemins solutions en des temps très faibles. Ainsi, l'étape de lissage nécessaire pour améliorer la qualité des chemins calculés s'avère être en pratique l'étape coûteuse limitant la performance du planificateur. Deux types d'approches peuvent être envisagées pour améliorer la qualité des chemins en conservant de bonnes performances. La première consiste à améliorer les trajectoires lors de leur exécution. Cette méthode permet de ne pas pénaliser la performance de la planifica-

tion en étalant les calculs d'optimisation sur une plage de temps plus importante pendant que le mouvement du robot s'exécute. Au contraire, la deuxième méthode consiste à travailler en amont de la planification, en modifiant les réseaux initiaux afin d'améliorer leur qualité. Cette extension pourrait notamment bénéficier de certains travaux allant déjà dans cette direction [Nieuwenhuisen 04a].

Vers un chemin globalement optimal. Comme dit précédemment, les réseaux de rétraction ouvrent la voie à une optimisation globale des chemins solutions. D'abord, la non redondance des cycles du réseau permet d'envisager pour une requête donnée, l'optimisation de l'ensemble des chemins solutions pouvant être extraits du réseau, puis de sélectionner parmi les chemins optimisés, de nouveau le plus optimal. De plus la capture des classes de chemins déformables à travers la notion de rétraction permet de suggérer des outils d'optimisation plus puissants, exploitant la nature des réseaux de rétraction. En particulier, on sait que le chemin optimal, comme tous les chemins de l'espace, peut être rétracté sur le réseau de rétraction (et ce quel que soit le critère d'optimalité fixé) et donc que réciproquement, il existe un chemin du réseau rétractable sur le chemin optimal. Ainsi, une voie de recherche intéressante concerne le développement d'un outil d'optimisation basé sur la notion de rétraction et "explorant", à partir d'un chemin donné, l'espace des chemins sur lesquels il est possible de se rétracter, à la recherche d'un chemin optimal dans cet espace. Au final, la combinaison de cet outil avec les réseaux de rétractions permettrait de garantir la convergence vers un chemin globalement optimal.

De part la généricité des techniques décrites dans cette thèse, ces travaux concernent des domaines applicatifs variés tels que la robotique et la réalité virtuelle. La prise en compte de scènes dynamiques dans les problèmes de planification de robots ou d'entités virtuelles vise une plus grande autonomie de mouvement grâce à l'intégration de contraintes environnementales plus réalistes. En robotique, la gestion des scènes dynamiques intervient également dans des problèmes où le robot en interaction avec l'environnement contribue à modifier la scène par ses propres actions. En particulier dans des tâches de manipulation d'objets, les phases de transit et de transfert peuvent être vues comme des environnements partiellement dynamiques où les obstacles mobiles correspondent aux objets déplaçables non transportés. Ainsi, les problèmes de planification de tâches de manipulation [Alami 89, Siméon 03], incluant notamment des problèmes plus spécifiques de navigation de robots mobiles au sein d'objets déplacables [Stiman 04], ou combinant la planification de mouvement à de la planification de tâches [Gravot 03] correspondent à des situations d'interaction qui peuvent bénéficier des méthodes présentées dans ce manuscrit. Dans un contexte plus large de prise en compte des interactions homme-robot [Chatila 02], les méthodes développées peuvent être étendues pour le calcul du mouvement possible de robots dans des scènes où les humains jouent le rôle des "obstacles mobiles".

Annexe A

Capture de l'Espace Libre par un Réseau

Nous présentons ici quelques notions principalement issues de la topologie algébrique afin de définir certains éléments caractéristiques d'un espace topologique (dans notre cas, l'espace des configurations). Ces informations placées dans le contexte de la planification de mouvement, permettront de mieux spécifier comment la structure des réseaux probabilistes peut refléter la nature d'un tel espace, notamment à travers les notions de *couverture*, de *connexité* et de *classes d'homotopie*.

Les éléments présentés proviennent essentiellement de [Massey 67] et [Latombe 91] (chap.2, sect.5). Ainsi le lecteur pourra se référer à ces ouvrages pour des informations complémentaires.

1 Couverture

Soit un ensemble de nœuds, plongé au sein de l'espace des configurations libres $\mathcal{C}_{free}$. Cet ensemble constitue une couverture de $\mathcal{C}_{free}$ si l'union des domaines de visibilité des nœuds couvre $\mathcal{C}_{free}$. Notons que l'existence d'une telle couverture dépend à la fois de la forme de $\mathcal{C}_{free}$ et de la méthode locale à partir de laquelle s'exprime la notion de visibilité. Une telle couverture est garantie d'exister seulement pour les espaces vérifiant une propriété que l'on appelle l'*ε-goodness* introduite dans [Kavraki 96]. On parle alors d'un espace $\mathcal{C}_{free}^{\varepsilon-good}$.

2 Composantes connexes

Un espace topologique est dit *connexe par arcs* si deux points quelconques peuvent toujours être reliés par un chemin. L'espace des configurations est un exemple particulier d'espace connexe par arc. L'espace des configurations libres $\mathcal{C}_{free}$ est quant à lui composé d'un ensemble

de sous-espaces connexes par arcs. On appelle alors ces espaces les *composantes connexes* de $\mathcal{C}_{free}$.

Ainsi, on dit qu'un réseau capture les différentes composantes connexes de l'espace, quand sa connexité est similaire à celle de l'espace libre.

3 Opérations sur les chemins

Soient τ et τ' deux chemins de l'espace topologique donné X, tels que $\tau(1) = \tau'(0)$. Le chemin $\tau \bullet \tau'$ défini par :

$$\tau \bullet \tau'(s) = \begin{cases} \tau(2s) & si \quad 0 \leq s \leq \frac{1}{2}, \\ \tau'(2s-1) & si \quad \frac{1}{2} \leq s \leq 1, \end{cases}$$

est appelé le *produit* (ou la composition) de τ par τ'.

On appelle *chemin nul* en q_0 le chemin ε_{q_0} tel que $\varepsilon_{q_0}(s) = q_0$ pour tout $s \in [0,1]$.

Soit τ un chemin. On appelle *chemin inverse* de τ, le chemin $\overline{\tau}$ tel que :
$\overline{\tau}(s) = \tau(1-s)$ pour tout $s \in [0,1]$.

4 Relation d'homotopie

Pour un espace topologique donné X, deux chemins τ et τ' ayant les points extrémaux confondus sont dits *homotopes*, si et seulement si il existe une transformation continue permettant de passer de l'un à l'autre. Autrement dit, s'il existe une fonction continue $H : [0,1] \times [0,1] \rightarrow X$ telle que :

$$\forall x \in [0,1] \begin{cases} H(x,0) = \tau(x), \\ H(x,1) = \tau'(x), \end{cases}$$

L'homotopie détermine une relation d'équivalence notée $\sim$ au sein des chemins partageant les mêmes extrémités. Si on a $\tau_1 \sim \tau_1'$, $\tau_2 \sim \tau_2'$ et $\tau_1(1) = \tau_2(0)$, alors on en déduit $\tau_1 \bullet \tau_2 \sim \tau_1' \bullet \tau_2'$. Etant donnés deux points connexes de X, la relation d'homotopie permet de définir leurs *classes d'homotopie* comme étant les différentes familles de chemins homotopes reliant ces deux points.

5 Connexité simple et multiple

Soit un espace topologique connexe par arcs. Cet espace est dit *simplement connexe* si pour tout couple de chemins τ, τ' de cet espace vérifiant $\tau(0) = \tau'(0)$ et $\tau(1) = \tau'(1)$, on a τ, τ' homotopes. Sinon, on dit que l'espace est *à connexité multiple*. L'outil utilisé pour caractériser le degré de connexité d'un espace topologique est ce que l'on appelle le *groupe fondamental*, présenté ci-dessous.

6 Groupe fondamental et classes d'homotopie

Soit x_0 un point arbitraire de X. Un *lacet* d'origine x_0 se définit comme un chemin λ de X tel que $\lambda(0) = \lambda(1) = x_0$. Soient $\Omega(X, x_0)$ l'ensemble de tous les lacets de X ayant pour origine x_0 et $\pi_1(X, x_0)$, l'ensemble de toutes les classes d'homotopies des lacets d'origine x_0. Si λ est un lacet de $\Omega(X, x_0)$, $[\lambda]$ définit la classe d'homotopie de $\pi_1(X, x_0)$ auquel appartient le lacet.

L'opération $[\lambda] \bullet [\lambda] = [\lambda \bullet \lambda]$ permet de munir $\pi_1(X, x_0)$ d'une structure de groupe. L'élément identité, appelé l'élément d'homotopie nulle correspond à $[\varepsilon_{x_0}]$, avec ε_{x_0}, chemin nul situé en x_0. L'inverse de tout $[\lambda] \in \pi_1(X, x_0)$ est $[\bar{\lambda}]$. Si X est connexe par arcs, les groupes $\pi_1(X, x_0)$ et $\pi_1(X, x_1)$ sont isomorphiques pour tout couple de points x_0, x_1 de X. Ainsi, on utilise plus généralement la notation $\pi_1(X)$ pour désigner n'importe lequel de ces groupes. Le groupe $\pi_1(X)$ est appelé le *groupe fondamental* de X.

On peut vérifier que le nombre de classes d'homotopies joignant toute paire de points quelconque est égale à la cardinalité du groupe fondamental de l'espace topologique en question. En particulier, un espace topologique connexe par arcs est simplement connexe si et seulement si son groupe fondamental ne contient comme seul élément que l'élément homotopique nul. Sinon, l'espace est à connexité multiple.

Ainsi, dans le domaine de la planification, construire un réseau qui capture les différentes classes d'homotopie de l'espace des configurations libres $\mathcal{C}_{free}$ signifie construire un réseau comprenant l'ensemble des lacets permettant de générer le groupe fondamental associé à l'espace libre considéré.

Annexe B

Calcul de Boules Libres de $\mathcal{C}$

Cette annexe présente une méthode de calcul de boules libres de $\mathcal{C}$ utilisée lors de la construction des réseaux de rétraction (c.f. chapitre 4 section 4.2). On suppose que le robot est constitué de n solides $S_1,...S_i,...S_n$ et de n liaisons cinématiques $J_0,...J_i,...J_{n-1}$, avec pour tout $i \in [1,n-1]$, S_i connecté à S_{i+1} par l'intermédiaire de J_i et J_0, liaison cinématique virtuelle entre S_0 et la scène. A chaque solide S_i est associé un repère R_i et R_0 représente un repère absolu fixe dans la scène. Pour une configuration q, on suppose également connus grâce au détecteur de collisions, les distances $l_1,...l_i,...l_n$, telles que pour tout $i \in [\![1,n]\!]$, l_i est une distance minimale de S_i aux obstacles dans l'espace de travail.

L'approche utilisée consiste à exprimer à partir des vitesses relatives entre les différents solides S_i, le temps minimum t_{safe} pour lequel ces solides ne peuvent rentrer en collision avec les obstacles. En prenant des vitesses relatives bien choisies il est ensuite possible d'établir une relation entre le temps t_{safe} obtenu à partir de ces vitesses relatives et le rayon r d'une boule libre de l'espace des configurations.

Torseurs cinématiques absolus

Les lois de composition des vitesses s'expriment de la façon suivante :

$$\begin{cases} \overrightarrow{V^{R_i/R_0}} = \overrightarrow{V^{R_{i-1}/R_0}} + \overrightarrow{\Omega^{R_{i-1}/R_0}} \wedge \overrightarrow{d^{R_{i-1}/R_i}} + \overrightarrow{V^{R_i/R_{i-1}}} \\ \overrightarrow{\Omega^{R_i/R_0}} = \overrightarrow{\Omega^{R_{i-1}/R_0}} + \overrightarrow{\Omega^{R_i/R_{i-1}}} \end{cases}$$

avec $\overrightarrow{d^{R_{i-1},R_i}}$ distance orientée séparant les origines des repères R_{i-1} et R_i. En passant à la norme de ces vitesses, on obtient :

$$\begin{cases} V^{R_i/R_0} \leq V^{R_{i-1}/R_0} + \Omega^{R_{i-1}/R_0}.d^{R_{i-1}/R_i} + V^{R_i/R_{i-1}} \\ \Omega^{R_i/R_0} \leq \Omega^{R_{i-1}/R_0} + \Omega^{R_i/R_{i-1}} \end{cases}$$

A partir de ces relations, il est donc possible de majorer pour un solide S_i les vitesses absolues $\begin{pmatrix} V \\ \Omega \end{pmatrix}^{R_i/R_0}$, à partir des vitesses relatives $\begin{pmatrix} V \\ \Omega \end{pmatrix}^{R_j/R_{j-1}}$ $j \in [\![1,i]\!]$ de l'ensemble des solides solides qui le précèdent.

Majoration des vitesses maximales

Les vitesses absolues associées aux différents S_i permettent alors de majorer les vitesses maximales absolues des points appartenant à chacun des S_i :

$$V^{R_0}_{M\in S_i} \leq V^{R_i/R_0} + \Omega^{R_i/R_0}.d_i^{max} = V_{S_i}^{max/R_0}$$

avec d_i^{max}, distance maximale d'un point de S_i à l'axe de rotation relative entre S_{i-1} et S_i.

Temps assurant S_i sans collision

La vitesse maximale d'un point de S_i mis en relation à la distance minimal de S_i aux obstacles est utilisée pour le calcul du temps t_{S_i} garantissant la non collision de S_i.

$$t_{S_i} = \frac{l_i}{V_{S_i}^{max/R_0}}$$

.

Temps sans collision

Enfin, le temps minimal sans collision est obtenu en prenant le minimum de tous les temps calculés précédemment :

$$t_{safe} = \min_i(t_{S_i})$$

Rayon d'une boule libre

Supposons que la norme utilisée est la norme 2, que les rotations des liaisons cinématiues J_i sont exprimées à l'aide des angles d'Euler et sont pondérées par rapport aux translations par des coefficients w_i. Une configuration q' située à une distance r de q vérifie :

$$||q-q'||^2 = r^2 = \sum_{i=1}^{n}\sum_{j=1}^{m_i}(x_j^i - x_j'^i)^2 + \sum_{i=1}^{n} w_k^2 \sum_{k=1}^{p_i}(\theta_k^i - \theta_k'^i)^2$$

avec :

$$\begin{cases} x_i^j & : \quad j\text{-ème degré de liberté en translation du joint } i \\ \theta_i^j & : \quad j\text{-ème degré de liberté en rotation du joint } i \\ m_i & : \quad \text{nombre de degrés de liberté en translation du joint } i \\ p_i & : \quad \text{nombre de degrés de liberté en rotation du joint } i \end{cases}$$

En utilisant les relations suivantes :

$$(\Delta X_i)^2 = \sum_{j=1}^{m_i}(x_j^i - x_j'^i)^2 \quad \text{et} \quad (\Delta\Theta_i)^2 \leq \sum_{k=1}^{p_i}(\theta_k^i - \theta_k'^i)^2$$

on obtient :

$$r^2 \geq \sum_{i=1}^{n}(\Delta X_i)^2 + \sum_{i=1}^{n}(w_i\Delta\Theta_i)^2$$

qui peut s'écrire à partir des vitesses relatives :

$$r^2 \geq t^2\left(\sum_{i=1}^{n}(V^{R_i/R_{i-1}})^2 + \sum_{i=1}^{n}(w_i\Omega^{R_i/R_{i-1}})^2\right)$$

Le temps t_{safe} présenté plus haut dépend des différentes vitesses relatives $\begin{pmatrix} V \\ \Omega \end{pmatrix}^{R_i/R_{i-1}}$. En prenant pour le calcul de ce temps, les vitesses relatives suivantes :

$$\begin{pmatrix} V \\ \Omega \end{pmatrix}^{R_i/R_{i-1}} = \begin{cases} \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \text{si la liaison est une pure translation} \\ \begin{pmatrix} 0 \\ \frac{1}{w_i} \end{pmatrix} & \text{si la liaison est une pure rotation} \\ \begin{pmatrix} 1 \\ \frac{1}{w_i} \end{pmatrix} & \text{si la liaison combine translations et rotations} \end{cases}$$

on obtient une relation liant t et r :

$$r^2 \geq t^2.m$$

où m correspond à la somme des liaisons de rotation et de translation apparaissant dans les joints ($m \leq 2i$).

Alors, en prenant $r_{safe} = t_{safe}\sqrt{m}$, on a la garantie d'avoir $t \leq \frac{r_{safe}}{\sqrt{m}} = t_{safe}$. Donc la boule de rayon r_{safe} centrée sur q est une boule libre de l'espace des configurations.

Cas du robot rigide

Considérons par exemple le cas d'un robot rigide à 6 degrés de liberté (3 translations, 3 rotations). On fixe le vecteur vitesse du robot de la façon suivante :

$$\begin{pmatrix} V \\ \Omega \end{pmatrix}^{R_1/R_0} = \begin{pmatrix} 1 \\ \frac{1}{w_1} \end{pmatrix}$$

ce qui permet d'obtenir une majoration de la vitesse maximale des points du robot :

$$V_M \leq V_0 + \Omega_0 . d_1^{max} = 1 + \frac{d_0^{max}}{w_1}$$

d'où le temps minimal sans collision :

$$t_{safe} = \frac{l_1}{V_{S_1^{max}/R_0}} = \frac{l_1}{1 + \frac{d_0^{max}}{w_1}}$$

et le rayon d'une boule libre de C centrée sur q :

$$r_{safe} = t_{safe} . \sqrt{2} = \frac{l_1 . \sqrt{2}}{1 + \frac{d_0^{max}}{w_1}}$$

Références bibliographiques

[Alami 89] R. Alami, T. Siméon & J.-P. Laumond. *A Geometrical Approach to Planning Manipulation Tasks. The Case of Discrete Placements and Grasps.* Proc. 5th Int. Symp. on Robotics Research, pages 453–463, 1989.

[Amato 98] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones & D. Vallejo. *OBPRM : An Obstacle-Based PRM for 3D Workspaces*. In P. Agarwal, L.E. Kavraki & M. Mason, editeurs, Robotics : The Algorithmic Perspective (WAFR1998), pages 155–168. A.K. Peters, 1998.

[Atramentov 02] A. Atramentov & S.M. LaValle. *Efficient Nearest Neighbor Searching for Motion Planning*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 632–637, 2002.

[Barraquand 91a] J. Barraquand & J.-C. Latombe. *Robot Motion Planning : A Distributed Representation Approach.* Int. J. Robot. Res., vol. 10, no. 6, pages 628–649, December 1991.

[Barraquand 91b] J. Barraquand & J.-C. Latombe. *Robot Motion Planning : A Distributed Representation Approach*. International Journal of Robotics Research, vol. 10(6), pages 628–649, 1991.

[Basu 00] S. Basu, R. Pollack & M.-F. Roy. *Computing Roadmaps of Semi-algebraic Sets on a Variety*. Journal of the American Mathematical Society, vol. 13(1), pages 55–82, 2000.

[Bessière 93] P. Bessière, J.M. Ahuactzin, E.-G. Talbi & E. Mazer. *The "Ariadne's Clew" Algorithm : Global Planning with Local Methods*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 1373–1380, 1993.

[Bohlin 00] R. Bohlin & L.E. Kavraki. *Path Planning using Lazy PRM*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 521–528, 2000.

[Boor 99] V. Boor, M.H. Overmars & A.F. van der Stappen. *The Gaussian Sampling Strategy for Probabilistic Roadmap Planners*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 1018–1023, 1999.

[Borenstein 91] J. Borenstein & Y. Koren. *The vector field histogram - fast obstacle avoidance for mobile robots*. IEEE Transactions on Robotics and Automation, pages 278–288, 1991.

[Brock 99] O. Brock & Oussama Khatib. *High-Speed Navigation Using the Global Dynamic Window Approach*. Proceedings of the International Conference on Robotics and Automation, pages 341–346, 1999.

[Brock 00] O. Brock & O. Khatib. *Real time replanning in high-dimensional configuration spaces using sets of homotopic paths*. International Conference on Robotics and Automation, pages 550–555, 2000.

[Brock 01] O. Brock & L. Kavraki. *Decomposition-based motion planning : A framework for real-time motion planning in high-dimensional configuration spaces*. In Proc. ICRA, volume 2, pages 1469–1474, 2001.

[Bruce 02] J. Bruce & M. Veloso. *Real-ime randomized path planning for robot navigation*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 2383–2388, 2002.

[Burns 03] B. Burns & O. Brock. *Information Theoretic Construction of Probabilistic Roadmaps*. Proc. IEEE Int. Conf. on Intelligent Robots and Systems, 2003.

[Burns 05a] B. Burns & O. Brock. *Sampling-based motion planning using predictive models*. Proc. IEEE Int. Conf. on Robotics and Automation, 2005.

[Burns 05b] B. Burns & O. Brock. *Single-Query Entropy-Guided Path Planning*. Proc. IEEE Int. Conf. on Robotics and Automation, 2005.

[Burns 05c] B. Burns & O. Brock. *Toward Optimal Configuration Space Sampling*. Proceedings of Robotics : Science and Systems, 2005.

[Canny 88] J.F. Canny. The complexity of robot motion planning. MIT Press, 1988.

[Chatila 02] R. Chatila, R. Alami, T. Siméon, J. Pettre & L. Jaillet. *Safe, Reliable and Friendly Interaction between Humans and Humanoids*. 3rd IARP Int. Workshop on Humanoid and Human Friendly Robotics, pages 83–87, 2002.

[Cortés 03] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanisms*. PhD thesis, Institut National Polytechnique de Toulouse, 2003.

[Cortés 04] J. Cortés & T. Siméon. *Sampling-Based Motion Planning under Kinematic Loop Closure Constraints*. Proc. Workshop on the Algorithmic Foundations of Robotics, 2004.

[Craig 89] J. J. Craig. Introduction to robotics. Addison-Wesley, Reading, MA, 1989.

[Donald 93] B.R. Donald, P.G. Xavier, J.F. Canny & J.H. Reif. *Kinodynamic Motion Planning*. Journal of the ACM, vol. 40(5), pages 1048–1066, 1993.

[Erdmann 87] M. Erdmann & T. Lozano-Pérez. *On multiple moving objects*. Algorithmica, vol. 2(4), pages 477–521, 1987.

[Esteves 06] C. Esteves, G. Arechavaleta, J. Pettre & J. P. Laumond. *Animation planning for virtual characters cooperation*. ACM Transactions on Graphics, 2006. To appear.

[Faverjon 84] B. Faverjon. *Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator*. In IEEE Int. Conf. Robot. & Autom., pages 504–512, 1984.

[Ferré 04] E. Ferré & J.P. Laumond. *An iterative diffusion method for part disassembly*. IEEE Int. Conf. on Robotics and Automation, 2004.

[Fox 97] D. Fox, W. Burgard & S. Thrun. *The dynamic window approach to collision avoidance*. IEEE Robotics and Automation magazine, vol. 4(1), pages 23–33, 1997.

[Fraichard 93] T. Fraichard. *Dynamic trajectory planning with dynamic constraints : A "state-time space" approach*. Proc. IEEE/RSJ International Conference on Robots and Systems, vol. 2, pages 1394–1558, 1993.

[Fraichard 99] T. Fraichard. *Trajectory Planning In a Dynamic Workspace : A 'State-Time' Approach*. Advanced Robotics, vol. 13(1), pages 75–94, 1999.

[Fujimua 95] K. Fujimua. *Time-minimum routes in time-dependent networks*. IEEE Transactions on Robotics and Automation, vol. 11(3), pages 343–351, 1995.

[Gayle 05] R. Gayle, W. Segars, M. C. Lin & D. Manocha. *Path Planning for Deformable Robots in Complex Environments*. Robotics : Systems and Science, 2005.

[Geraerts 04] R. Geraerts & M.H. Overmars. *On Improving the Path Quality for Motion Planning*. Conference of the Advanced School for Computing and Imaging, 2004.

[Gravot 03] F. Gravot, S. Cambon & R. Alami. *aSyMov : A Planner that Deals with Intricate Symbolic and Geometric Problems*. Proc. 11th Int. Symp. of Robotics Research, 2003.

[Gupta 98] K. Gupta & A.P. del Pobil. Practical motion planning in robotics. John Wiley & Sons, 1998.

[Han 01] L. Han & N.M. Amato. *A Kinematics-Based Probabilistic Roadmap Method for Closed Kinematic Chains*. In B.R. Donald, K.M. Lynch

& D. Rus, editeurs, Algorithmic and Computational Robotics : New Directions (WAFR2000), pages 233–245. A.K. Peters, 2001.

[Hoffmann 89] C. M. Hoffmann. Geometric and solid modeling. Morgan Kaufmann, San Mateo, CA, 1989.

[Holleman 98] C. Holleman, L.E. Kavraki & J. Warren. *Planning Paths for a Flexible Surface Patch*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 21–26, 1998.

[Hopcroft 84] J.E. Hopcroft, J.T. Schwartz & M. Sharir. *On the Complexity of Motion Planning for Multiple Independent Objects : PSPACE-Hardness of the 'Warehouseman's Problem'*. Int. J. of Robotics Research, vol. 3(4), pages 76–88, 1984.

[Hsu 97] D. Hsu, J.-C. Latombe & R. Motwani. *Path Planning in Expansive Configuration Spaces*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 2719–2726, 1997.

[Hsu 98] D. Hsu, L.E. Kavraki, J.-C. Latombe, R. Motwani & S. Sorkin. *On Finding Narrow Passages with Probabilistic Roadmap Planners*. In P. Agarwal, L.E. Kavraki & M. Mason, editeurs, Robotics : The Algorithmic Perspective (WAFR1998), pages 141–153. A.K. Peters, 1998.

[Hsu 02] D. Hsu, R. Kindel, J.-C. Latombe & S. Rock. *Randomized Kinodynamic Motion Planning with Moving Obstacles*. International Journal of Robotics Research, vol. 21(3), pages 233–255, 2002.

[Hsu 03] D. Hsu, T. Jiang, J. Reif & Z. Sun. *The bridge test for sampling narrow passages with probabilistic roadmap planners*. Proc. IEEE Int. Conf. on Robotics and Automation, 2003.

[Huang 04] Y. Huang & K. Gupta. *A Delaunay triangulation Based Node Connection Strategy for Probabilistic Roadmap Planners*. Proc. IEEE International Conference on Robotics and Automation, pages 908– 913, 2004.

[Jaillet 04] L. Jaillet & T. Siméon. *A PRM-based Motion Planner for Dynamically Changing Environments*. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 1606–1611, 2004.

[Jaillet 05] L. Jaillet, A. Yershova, S. M. LaValle & T. Siméon. *Adaptive Tuning of the Sampling Domain for Dynamic-Domain RRTs*. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2005.

[Jiménez 98] P. Jiménez, F. Thomas & C. Torras. *Collision Detection Algorithms for Motion Planning*. In J.-P. Laumond, editeur, Robot Motion Planning and Control, pages 305–343. Springer-Verlag, 1998.

[Kant 86] M.H. Kant & P.A Whitelock. *Toward efficient trajectory planning : The path-velocity decomposition*. The International Journal of Robotics Research, vol. 5(3), pages 72–89, 1986.

[Kavraki 95] L.E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1995.

[Kavraki 96] L.E. Kavraki, P. Svestka, J.-C. Latombe & M.H. Overmars. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*. IEEE Transactions on Robotics and Automation, vol. 12(4), pages 566–580, 1996.

[Khatib 86] O. Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. International Journal of Robotics Research, vol. 5(1), pages 90–98, 1986.

[Khatib 97] O. Khatib, H. Jaouni, R. Chatila & J.P. Laumond. *Dynamic path modification for car-like non-holonomic mobile robots*. Proc. IEEE International Conference on Robotics and Automation, 1997.

[Ko 98] N.Y. Ko & R. Simmons. *The lane-curvature method for local obstacle avoidance*. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 1998.

[Kuffner 99] J.J. Kuffner. *Autonomous Agents for Real-Time Animation*. PhD thesis, Stanford University, 1999.

[Kuffner 00] J.J. Kuffner & S.M. LaValle. *RRT-Connect : An Efficient Approach to Single-Query Path Planning*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 995–1001, 2000.

[Lamiraux 97] F. Lamiraux. *Robots Mobiles à Remorque : De la Planification de Chemins à Exécution de Mouvements*. PhD thesis, Institut National Polytechnique de Toulouse, 1997.

[Lamiraux 01a] F. Lamiraux & L.E. Kavraki. *Planning Paths for Elastic Objects under Manipulation Constraints*. International Journal of Robotics Research, vol. 20(3), pages 188–208, 2001.

[Lamiraux 01b] F. Lamiraux & J.-P. Laumond. *Smooth Motion Planning for Car-Like Vehicles*. IEEE Transactions on Robotics and Automation, vol. 17(4), pages 498–501, 2001.

[Lamiraux 04] F. Lamiraux, D. Bonnafous & O. Lefebvre. *Reactive Path Deformation for Non-holonomic Mobile Robots*. IEEE Transactions on Robotics, vol. 20, no. 6, pages 967–977, December 2004.

[Large 03] F. Large. *Navigation Autonome d'un robot mobile en environnement dynamique et incertain*. PhD thesis, Université de Savoie, 2003.

[Large 04] F. Large, A.D. Vasquez, T. Fraichard & C. Laugier. *Avoiding Cars and Pedestrians using V-Obstacles and Motion Prediction*. In Proc. of the IEEE Intelligent Vehicle Symp., Pisa (IT), June 2004. Poster session.

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

[Latombe 99] J.-C. Latombe. *Motion Planning : A Journey of Robots, Molecules, Digital Actors, and Other Artifacts*. International Journal of Robotics Research, vol. 18(11), pages 1119–1128, 1999.

[Laumond 86] J.-P. Laumond. *Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints*. Proc. Int. Conf. on Intelligent Autonomous Systems, pages 346–354, 1986.

[Laumond 98a] J.-P. Laumond. Robot motion planning and control. Springer, 1998.

[Laumond 98b] J.-P. Laumond, S. Sekhavat & F. Lamiraux. *Guidelines in Nonholonomic Motion Planning for Mobile Robots*. In J.-P. Laumond, editeur, Robot Motion Planning and Control, pages 1–53. Springer-Verlag, 1998.

[Laumond 01] J.-P. Laumond & T. Siméon. *Notes on Visibility Roadmaps and Path Planning*. In B.R. Donald, K.M. Lynch & D. Rus, editeurs, Algorithmic and Computational Robotics : New Directions (WAFR2000), pages 317–328. A.K. Peters, 2001.

[LaValle 98] S.M. LaValle. *Rapidly-Exploring Random Trees : A New Tool for Path Planning*. TR 98-11, Computer Science Dept., Iowa State University, 1998.

[LaValle 99] S.M. LaValle, J.H. Yakey & L.E. Kavraki. *A Probabilistic Roadmap Approach for Systems with Closed Kinematic Chains*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 1671–1676, 1999.

[LaValle 01a] S.M. LaValle & P. Konkimalla. *Algorithms for Computing Numerical Optimal Feedback Motion Strategies*. International Journal of Robotics Research, vol. 20(9), pages 729–752, 2001.

[LaValle 01b] S.M. LaValle & J.J. Kuffner. *Rapidly-Exploring Random Trees : Progress and Prospects*. In B.R. Donald, K.M. Lynch & D. Rus, editeurs, Algorithmic and Computational Robotics : New Directions (WAFR2000), pages 293–308. A.K. Peters, 2001.

[LaValle 05] S.M. LaValle. Planning algorithms. Cambridge University Press, 2004-2005. To appear. Available at `http ://msl.cs.uiuc.edu/planning/`.

[Lean 96a] A. Mc Lean & C. Laugier. *Update and Repair of a Roadmap after Model Error Discovery*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, vol. 2, pages 917–924, 1996.

[Lean 96b] A. Mc Lean & I. Mazon. *Incremental roadmaps and global path planning in evolving industrial environments*. Proc. IEEE Int. Conf. on Robotics and Automation, vol. 2, pages 917–924, 1996.

[Leven 00] P. Leven & S. Hutchinson. *Toward Real-Time Path Planning in Changing Environments*. Proc. Workshop on the Algorithmic Foundations of Robotics, 2000.

[Leven 02] P. Leven & S. Hutchinson. *A framework for real-time path planning In changing environments*. The International Journal of Robotics Research, vol. 21, pages 999–1030, 2002.

[Lien 03] J.-M. Lien, S.L. Thomas & N.M. Amato. *A General Framework for Sampling on the Medial Axis of the Free Space*. Proc. IEEE Int. Conf. on Robotics and Automation, 2003. In press.

[Lin 03] M. Lin & D. Manocha. *Collision and proximity queries*, 2003.

[Lozano-Pérez 87] T. Lozano-Pérez. *A Simple Motion-Planning Algorithm for General Robot Manipulators*. IEEE J. of Robot. & Autom., vol. RA-3, no. 3, pages 224–238, Jun 1987.

[Lozano-Pérez 83] T. Lozano-Pérez. *Spatial Planning : A Configuration Space Approach*. IEEE Transactions on Computers, vol. 32(2), pages 108–120, 1983.

[Massey 67] W.S. Massey. Algebraic topology : Introduction. Springer-Verlag, 1967.

[Minguez 00] J. Minguez & L. Montano. *Nearness diagram navigation*. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2000.

[Minguez 02] J. Minguez, L. Montano, T. Siméon & R. Alami. *Global nearness diagram navigation*. IEEE Int. Conf. on Robotics and Automation, 2002.

[Mortenson 85] M. E. Mortenson. Geometric modeling. Wiley, New York, NY, 1985.

[Nielsen 00] C.L. Nielsen & L.E. Kavraki. *A Two Level Fuzzy PRM for Manipulation Planning*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 1716–1722, 2000.

[Nieuwenhuisen 04a] D. Nieuwenhuisen, A. Kamphuis & M.H. Overmars. *Automatic Construction of High Quality Rodmaps for Path Planning*. Proc. International Conference on Computer Games, Artificial Intelligence, Design and Education, 2004.

[Nieuwenhuisen 04b] D. Nieuwenhuisen & M.H. Overmars. *Useful Cycles in Probabilistic Roadmap Graphs*. Proc. IEEE International Conference on Robotics and Automation, pages 446–452, 2004.

[Nissoux 99] C. Nissoux. *Visibilité et Méthodes Probabilistes pour la Planification de Mouvement en Robotique*. PhD thesis, Université Paul Sabatier, 1999.

[O'Donnel 89] P.. O'Donnel. *Deadlock-Free and Collision Free Coordination of Two Robot Manipulators*. IEEE Int. Conf. on Robotics and Automation, pages 484–489, 1989.

[Overmars 95] M.H. Overmars & P. Svestka. *A Probabilistic Learning Approach to Motion Planning*. In K. Goldberg, D. Halperin, J.-C. Latombe & R. Wilsonet, editeurs, Algorithmic Foundations of Robotics (WAFR1994), pages 19–37. A.K. Peters, 1995.

[Paul 81] R. P. Paul. Robot manipulators : Mathematics, programming, and control. MIT Press, Cambridge, MA, 1981.

[Petti 05] S. Petti & T. Fraichard. *Partial Motion Planning Framework for Reactive Planning Within Dynamic Environments*. In Proc. of the IFAC/AAAI Int. Conf. on Informatics in Control, Automation and Robotics, Barcelona (SP), September 2005.

[Pettre 02] J. Pettre, T. Siméon & J.-P. Laumond. *Planning Human Walk in Virtual Environments*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 3048–3053, 2002.

[Quinlan 93] S. Quinlan & O. Khatib. *Elastic Bands : connecting path planning and control*. Proc. IEEE International Conference on Robotics and Automation, pages 802–807, 1993.

[Reif 85] J.H Reif & M. Sharir. *Motion planning in the presence of moving obstacles*. IEEE Symp. on Foundat. of Comp. Sci., pages 144–154, 1985.

[Schmitzberger 02] E. Schmitzberger, J.L. Bouchet, M. Dufaut, W. Didier & R. Husson. *Capture of homotopy classes with probabilistic road map*. IEEE/RSJ Int. Conf. on Robots and Systems, 2002.

[Schwartz 83a] J. T. Schwartz & M. Sharir. *On the Piano Movers' Problem : II. General Techniques for Computing Topological Properties of Algebraic Manifolds*. Communications on Pure and Applied Mathematics, vol. 36, pages 345–398, 1983.

[Schwartz 83b] J.T. Schwartz & M. Sharir. *On the Piano Movers' Problem II : General Techniques for Computing Topological Properties of Real Algebraic Manifolds*. Advances in Applied Mathematics, vol. 4, pages 298–351, 1983.

[Schwartz 83c] J.T. Schwartz & M. Sharir. *On the Piano Movers' Problem : III. Coordinating the motion of several independent bodies*. Int. Journal of Robotics Research, vol. 2(3), pages 97–140, 1983.

[Sekhavat 98] S. Sekhavat, P. Svestka J.-P. Laumond & M.H. Overmars. *Multi-Level Path Planning for Nonholonomic Robots using Semi-Holonomic Subsystems*. International Journal of Robotics Research, vol. 17(8), pages 840–857, 1998.

[Shiller 96] P. Fiorini Z. Shiller. *Time optimal trajectory planning in dynamic environments*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 1553–1558, 1996.

[Simmons 96] R. Simmons. *The curvature-velocity method for local obstacle avoidance*. IEEE Int. Conf. on Robotics and Automation, 1996.

[Siméon 00] T. Siméon, J.-P. Laumond & C. Nissoux. *Visibility-Based Probabilistic Roadmaps for Motion Planning*. Advanced Robotics Journal, vol. 14(6), pages 477–494, 2000.

[Siméon 01] T. Siméon, J.-P. Laumond & F. Lamiraux. *Move3D : a Generic Platform for Path Planning*. Proc. IEEE Int. Symp. on Assembly and Task Planning, pages 25–30, 2001.

[Siméon 02] T. Siméon, S. Leroy & J.-P. Laumond. *Path coordination for multiple mobile robots : a resolution complete algorithm*. IEEE Transaction on Robotics and Automation, vol. 18(1), 2002.

[Siméon 03] T. Siméon, J.-P. Laumond, J. Cortés & A. Sahbani. *Manipulation Planning with Probabilistic Roadmaps*. International Journal of Robotics Research, 2003.

[Stiman 04] M. Stiman & J. J. Kuffner. *Navigation among movable obstacles : Real-time reasoning in complex environments*. Proc. IEEE Int. Conf. on Humanoid Robotics (Humanoids'04), 2004.

[Svestka 97a] P. Svestka. *Robot Motion Planning using Probabilistic Roadmaps*. PhD thesis, Universiteit Utrecht, 1997.

[Svestka 97b] P. Svestka & M.H. Overmars. *Motion Planning for Car-like Robots, a Probabilistic Learning Approach*. International Journal of Robotics Research, vol. 16(2), pages 119–143, 1997.

[Sánchez 03] G. Sánchez & J.-C. Latombe. *A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking*. In R. Jarvis & A. Zelinsky, editeurs, Robotics Research, the Tenth International Symposium, pages 403–417. Springer Tracts in Advanced Robotics, Springer, 2003.

[Ulrich 98] I. Ulrich & J. Borenstein. *VFH+ : Reliable obstacle avoidance for fast mobile robots*. IEEE Int. Conf. on Robotics and Automation, 1998.

[Ulrich 00] I. Ulrich & J. Borenstein. *VFH* : Local obstacle avoidance with lookahead verification*. IEEE Int. Conf. on Robotics and Automation, 2000.

[van den Berg 04] J.P. van den Berg & M.H. Overmars. *Roadmap-based motion planning in dynamic environments*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pages 1598–1605, 2004.

[van den Berg 05a] J.P. van den Berg, D. Nieuwenhuisen, L. Jaillet & M.H. Overmars. *Creating Robust Toadmaps for Motion Planning in Changing Environments*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2005.

[van den Berg 05b] J.P. van den Berg & M.H. Overmars. *Prioritized Motion Planning for Multiple Robots*. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2005.

[van den Bergen 04] G. van den Bergen. Collision detection in interactive 3d environments. Morgan Kaufmann Publishers, 2004.

[van Geem 01] C. van Geem, T. Siméon & J. Cortés. *Progress Report on Collision Detection*. In Second Year Deliverables of the MOLOG Project. Rapport LAAS, 2001.

[Warren 90] C.W. Warren. *Multiple Robot Path Coordination Using Artificial Potential Field*. IEEE Int. Conf. on Robotics and Automation, vol. 1, pages 500–505, 1990.

[Wilmarth 99] S. Wilmarth, N.M. Amato & P.Stiller. *MAPRM : A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space*. Proc. IEEE Int. Conf. on Robotics and Automation, pages 1024–1031, 1999.

[Yershova 05] A. Yershova, L. Jaillet, T. Siméon & S. M. LaValle. *Dynamic-Domain RRTs : Efficient Exploration by Controlling the Sampling Domain*. IEEE Int. Conf. on Robotics and Automation, 2005.

RÉSUMÉ :

Malgré le franc succès des techniques de planification de mouvement au cours de ces deux dernières décennies, leur adaptation à des scènes comprenant à la fois des obstacles statiques et des obstacles mobiles s'est avérée limitée jusqu'ici. Une des raisons en est le coût associé à la mise à jour des structures de données précalculées afin de capturer la connexité de l'espace libre. Notre contribution principale concerne la proposition d'un nouveau planificateur capable de traiter ces problèmes d'environnements partiellement dynamiques composés à la fois d'obstacles statiques et d'obstacles mobiles.

Le chapitre 2 expose le principe général d'un planificateur dédié aux environnements à changement dynamique. La stratégie proposée se base sur la combinaison des techniques de type réseaux PRM avec des méthodes de diffusion. Elle utilise aussi des mécanismes de mise à jour paresseux permettant de rendre l'approche particulièrement efficace. Le chapitre 3 présente une méthode originale de diffusion appelée RRT à Domaine Dynamique. Celle-ci peut en particulier servir à reconnecter les portions de structure de données localement invalidées par les obstacles dynamiques. Les chapitres 4 et 5 présentent deux méthodes de création de réseaux cycliques qui servent à initialiser le planificateur. La première assume que les obstacles mobiles sont confinés dans une région donnée, pour construire un réseau adapté aux différents types de changements de position possibles. La seconde est une méthode qui construit des réseaux appelées "réseaux de rétraction". A l'aide d'une structure de donnée de faible taille, cette structure parvient à capturer les différentes variétés de chemins de l'espace, à travers notamment chacune des classes d'homotopie de l'espace libre.

SUMMARY :

Recently, new motion planning techniques have proved to be very efficient. Nevertheless, there adaptation to scenes with both static and mobile obstacles has been limited so far. One of the reasons is the cost of the dynamic data structures updates during the dynamic changes of the environments. Our main contribution is the proposition of a new planner able to deal with these partially dynamic environments.

The second chapter shows the general principle of a planner dedicated to environments with both static and mobile obstacles. This hybrid planner combines the PRM framework with improved diffusion techniques. Several lazy evaluation mechanisms are also used to improve the global efficiency. The chapter 3 presents the Dynamic-Domain RRT planner, an efficient diffusion technique which can be used to locally repair the broken portions of the roadmap. The chapters 4 and 5 present two methods to initialize the planner with create cyclic roadmaps. The first one uses the assumption that mobile obstacles are confined to some regions of the space to build roadmap specifically adapted to obstacles positions changes. The second one is a general method to capture in a compact data structure the different path varieties of the free configuration space.