Table des matières

1 Introduction

Aujourd'hui, la réalité virtuelle et l'animation graphique sont de plus en plus omniprésentes : les mondes virtuels en ligne, les jeux vidéos, les films d'animation. Dans la plupart de ces domaines l'humain virtuel est au centre de l'attention. Modéliser un personnage et son comportement de manière réaliste est devenu un des problèmes principaux de l'animation graphique. La génération d'un comportement réaliste pour un personnage virtuel, de ses gestes à son mode de locomotion, peut être traitée comme un problème de planification de mouvement.

Bien qu'ayant eu la plupart de ses contributions par le biais de la robotique et des roboticiens, la planification de mouvement recouvre plusieurs domaines, des plus généraux aux plus exotiques. En effet, qui n'a pas eu à déménager une armoire, un frigo, ou d'autres meubles encombrants en passant par des escaliers sinueux, ou des couloirs étroits. Ce type de problèmes relève de la planification de mouvement. Aujourd'hui, les champs d'application de ce domaine sont très variés : pliage de protéines, assemblage de véhicules, simulation et animation de foules.

L'animation de personnages virtuels est un problème récurrent. Les méthodes actuelles demandent beaucoup de temps et d'expérience. De nombreux efforts sont portés sur la facilitation de ce travail. Un des moyens est de donner plus d'autonomie aux personnages virtuels, leur permettre de faire des gestes et des déplacements sans demande explicite de l'opérateur. Mais cela diminue le contrôle de l'utilisateur sur le personnage. Cette thèse propose, à travers la planification de mouvement, de trouver un compromis entre autonomie du personnage virtuel et contrôle de l'opérateur en développant une méthode de coopération entre humain et machine dans le cadre de la recherche de chemin et de l'animation de personnages

virtuels.

1.1 Contribution

Les principales contributions de cette thèse sont d'une part une méthode de planification de mouvement dédiée aux mouvements dans des espaces des configurations faiblement connexes. Une étude détaillée des performances de la méthode sur les environnements du type considéré confirme la validité et les avantages de l'approche. Bien que la méthode et l'étude aient été effectuées d'une manière générique sur ce type d'environnement, l'objectif reste le déplacement dans des environnements « humains » structurés (intérieurs de bâtiments, de maisons, de bureaux), qui font partie du type d'environnements étudiés.

D'autre part, une nouvelle méthode de planification de mouvement qui permet une coopération efficace entre un algorithme et un opérateur humain lors d'une recherche de chemin est présentée. Cette méthode permet non seulement de trouver un chemin solution rapidement de manière interactive, mais aussi de respecter des contraintes imposées par l'opérateur sur le type de solution en combinant les avantages d'une recherche manuelle et d'une recherche automatique. Là où les précédents travaux proposent des solutions découplées en plusieurs phases, notre contribution est d'intégrer l'utilisateur dans la boucle de planification et de le mettre au centre de la recherche en utilisant des périphériques spécifiques pour une meilleure immersion.

Enfin, notre dernière contribution concerne l'animation de personnages virtuels. En partant de travaux effectués précédemment, nous avons ajouté un contrôle de l'utilisateur sur la boucle d'animation de locomotion, en adaptant notre planificateur interactif au problème de la marche humaine. Cette méthode permet l'édition intuitive et interactive de trajectoires de locomotion humaine animées en se basant sur un nombre limité de captures de mouvement.

1.2 Organisation du document

Ce document est composé de 4 chapitres principaux qui sont organisés de la façon suivante :

- Le chapitre 2 introduit les concepts de base de la planification de mouvement et les travaux déjà effectués dans ce domaine, ainsi que les notions-clés utilisées tout au long de ce document. Après avoir décrit les travaux sur lesquels ce chapitre se base, un algorithme de planification de mouvement spécifique aux environnements faiblement connexes est présenté.
- Le chapitre 3 présente une méthode permettant la coopération entre un opérateur humain et un algorithme de planification de mouvement pour la recherche de trajectoire, à l'aide de périphériques d'interaction. Entre autres applications, un cas de désassemblage de

pièces mécaniques à l'aide d'un périphérique à retour d'effort illustre notre approche.
– Le chapitre 4 utilise la méthode introduite dans le chapitre 3 en association avec un contrôleur de locomotion basé sur l'interpolation de captures de mouvement pour animer un personnage virtuel de manière interactive. Deux versions différentes de ce planificateur de locomotion sont présentées, chacune avec des applications spécifiques.
– Le chapitre 5 conclue sur les travaux effectués durant cette thèse et suggère quelques perspectives.

1.3 Publications associées

1. David FLAVIGNÉ, Michel TAÏX, Étienne FERRÉ. **Interactive Motion Planning for Assembly tasks** . Dans *IEEE International Symposium on Robot and Human Interactive Communication (Ro-Man 2009)*, Toyama International Conference Center, Japan, 2009.
2. David FLAVIGNÉ, Michel TAÏX. **Méthode hybride de planification de trajectoires pour environnement mixte**. Dans *Reconnaissance des Formes et Intelligence Artificielle (RFIA 2010)*, Centre de Congrès, Caen, France, 2010.
3. David FLAVIGNÉ, Michel TAÏX. **Improving Motion Planning in Weakly Connected Configuration Spaces**. Dans *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, Taipei International Convention Center, Taipei, Taiwan, 2010.
4. Tan Viet Anh TRUONG, David FLAVIGNÉ, Julien PETTRÉ, Katja MOMBAUR, Jean-Paul LAUMOND. **Reactive synthesizing of human locomotion combining nonholonomic and holonomic behaviors**. Dans *IEEE BioRob2010*, Tokyo, Japan, 2010.

2 Planification de mouvement en environnement mixte

2.1 Introduction

Avant de développer la partie interactive de ces travaux, il nous a semblé nécessaire d'améliorer des algorithmes de base déjà existants dans le domaine de la planification de mouvement afin de les adapter à notre thématique. Ce travail est motivé par l'utilisation de ces algorithmes dans le cadre de l'animation graphique. Il s'agira de guider manuellement un personnage virtuel dans un environnement structuré avec l'aide d'un algorithme de planification de mouvement. Ce thème sera abordé dans le dernier chapitre de ce document. Dans ce chapitre, nous étudierons comment adapter un algorithme de base pour une planification dans un environnement structuré pour la locomotion humaine. Ce type d'environnement correspond en réalité à des environnements intérieurs (maisons, usines etc.). Ils sont souvent formés d'une succession de pièces séparées par des passages étroits (portes ou couloirs). C'est donc sur des environnements de ce style que nous nous concentrerons.

Malgré l'apparente simplicité de ces environnements, l'espace de travail en trois dimensions reste assez complexe et une exploration exhaustive par l'algorithme reste hors de question car cela demande trop de ressources et de temps de calcul. La possibilité d'avoir des environnements dynamiques obligeant l'algorithme à ré-explorer certaines zones accentue encore la complexité. Ce sont ces contraintes qui ont motivé les choix que nous avons faits

durant cette première partie. Dans un premier temps nous allons définir quelques concepts importants de la planification de mouvement, donner un bref historique du domaine de la planification de mouvement et rappeler les méthodes de base les plus connues. Ensuite nous décrirons les différents algorithmes sur lesquels nous nous sommes basés pour développer notre méthode, avant de décrire la méthode elle-même. Les résultats présentés dans la dernière partie, ainsi que leur analyse, permettront d'en montrer les avantages.

2.1.1 Planification de mouvement

2.1.1.1 Concepts

Dans cette section, nous allons décrire des concepts de planification de mouvement importants que nous utiliserons tout au long de ce mémoire.

- L'**espace de travail**, noté W, est l'espace dans lequel la géométrie de l'environnement et du robot est décrite.
- Le nombre de **degré de liberté**, noté ddl, indique le nombre de paramètres indépendants nécessaires pour caractériser les transformations à appliquer à un système pour le déplacer.
- Une **configuration** correspond à l'ensemble des valeurs de chaque ddl pour une situation précise du système. Elle caractérise le placement de tous les corps du système.
- L'**espace des configurations**, noté CS, est l'ensemble des configurations possibles pour le système. Le déplacement du système dans l'espace de travail correspond au déplacement d'un point dans l'espace des configurations..
- Une méthode de **détection de collision** permet de définir si le système est en collision avec un obstacle pour une configuration donnée. $Collision : CS \rightarrow 0,1$
- L'**espace des configurations libres**, noté CS_{free} est l'ensemble des configurations de CS qui ne sont pas en collision avec un obstacle de l'environnement. $\forall q_i \in CS : Collision(q_i) \rightarrow q_i \notin CS_{free}$
- La **méthode d'échantillonnage** permet de choisir une configuration appartenant à CS.
- Un **échantillon** est la configuration choisie par la méthode d'échantillonnage.
- Un **chemin** est une fonction $Chemin : [0\ 1] \rightarrow CS$ qui relie deux configurations de CS. L'ensemble des chemins de CS est noté C.
- La **méthode locale** est une méthode liée au système à déplacer qui permet de définir un chemin entre deux configurations de CS. $ML : CS \times CS \rightarrow C$
- Une **composante connexe** d'un graphe est une sous-partie CC d'un graphe G telle que $(\forall q_i \in CC) \wedge (\forall q_j \in CC) : \exists Chemin(q_i, q_j) \in G$

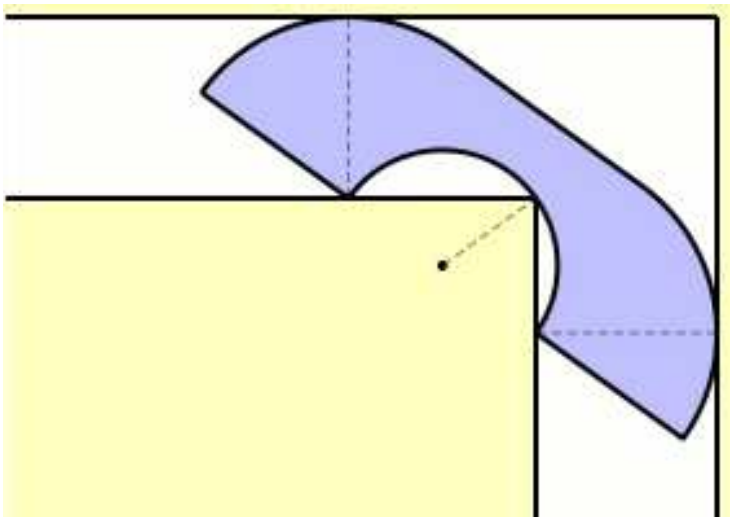


FIG. 2.1 – De [Esteves Jaramillo 2007]. Illustration du problème du sofa.

2.1.1.2 Le problème de la planification de mouvement

D'une manière générale, planifier un mouvement revient à ordonner les positions d'un corps dans l'espace afin de lui faire atteindre une position voulue. Pour un être humain, planifier un mouvement est une action naturelle. Il peut planifier facilement ses déplacements, ainsi que ceux des objets qu'il contrôle : voitures, bateaux, avions, etc. Mais dans certaines situations, cela devient un problème complexe qui nécessite beaucoup de temps et d'efforts. L'exemple le plus représentatif est celui du problème du sofa (illustré figure 2.1), posé par Leo Moser en 1966 [Moser 1966], qui revient à chercher la plus grande taille de sofa que l'on puisse déplacer dans un couloir à angle droit, de largeur unitaire. Ce type de problème est à l'origine des travaux dans le domaine de la planification de mouvement en robotique.

Plus tard, Schwarz et Sharir posent les bases du problème du déménageur de piano [Schwartz and Sharir 1987], qui consiste à déplacer un piano composé d'un seul bloc à travers une pièce encombrée d'obstacles (figure 2.2). Ce problème sera ensuite étendu par J.H. Reif comme le « problème général du déménageur ». Cette généralisation consiste à déplacer un objet quelconque, qui peut-être composé de plusieurs corps articulés entre eux (on peut prendre comme exemple une chaîne, ou un train composé de plusieurs wagons). Le déplacement d'un piano ou celui d'un corps articulé peuvent être vus comme deux problèmes totalement différents, bien qu'ils soient similaires. La question est de trouver une méthodologie permettant d'avoir une approche générique de ce type de problèmes.

T. Lozano-Perez propose une solution originale. Il s'agit de présenter le problème, non pas dans l'espace de travail W, mais dans un espace qui peut rendre compte de toutes les situations possibles du corps considéré. Par exemple, pour un corps articulé, chaque articulation

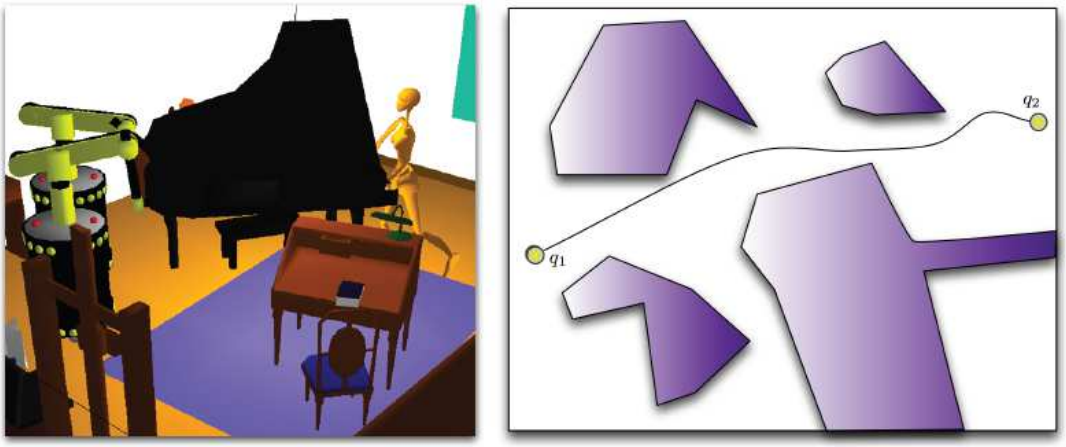


FIG. 2.2 – De [Esteves Jaramillo 2007]. Illustration du problème du déménageur de piano. À gauche un aperçu de l'espace de travail et à droite une représentation possible de l'espace des configurations. Le problème est de déplacer le piano de la configuration q_1 à la configuration q_2, sans entrer en collision avec les obstacles violets.

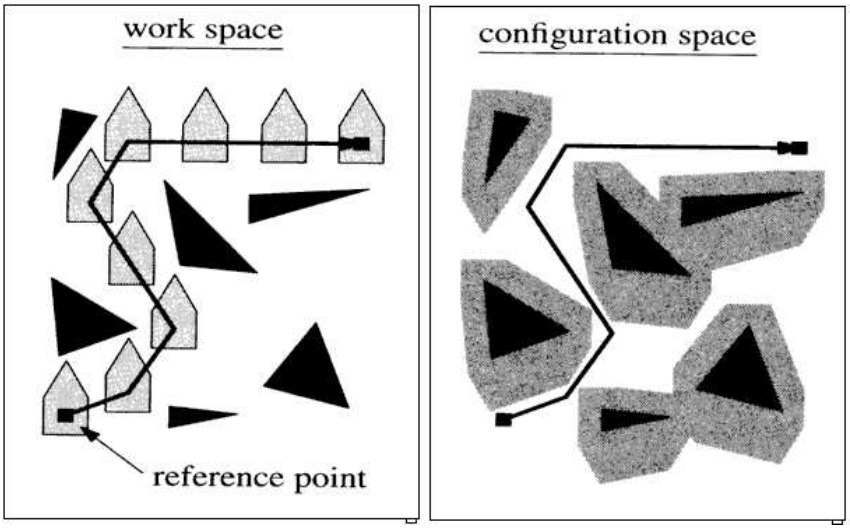


FIG. 2.3 – De [Esteves Jaramillo 2007]. Passage d'un espace de travail usuel (à gauche), vers l'espace des configurations correspondant (à droite).

peut représenter un ou plusieurs degrés de liberté. Un objet « volant » (c'est-à-dire qui peut se déplacer librement dans toutes les directions), peut bouger selon ses six degrés de liberté : trois pour les translations et trois pour les rotations. La position de cet objet serait donc décrite par six paramètres, chaque paramètre étant la valeur d'un des degrés de liberté. L'ensemble de ces paramètres est appelé une configuration. Le nombre de paramètres définissant la configuration de l'objet considéré correspond à la **dimension** de *CS*. Ainsi, une configuration dans *CS* définit un placement de l'objet dans *W*. Il est possible de décrire le problème général du déménageur comme étant la recherche d'un chemin dans l'espace des configurations libres entre une configuration initiale et une configuration finale. Le passage de l'espace de travail *W* vers l'espace des configurations *CS* est illustré pour un objet simple avec deux degrés de liberté en translation sur la figure 2.3. Le problème de déplacer un système (ensemble de corps) dans un espace de travail *W* revient à déplacer un point dans un *CS* de dimension *n*. Chaque configuration *q* de dimension *n* contient alors la situation de tous les corps du système. Depuis lors, ce problème a été transposé dans différents domaines allant de l'industrie à la biologie. Les différentes applications que nous pouvons trouver aujourd'hui à la planification de mouvement sont aussi variées que les bras robotiques utilisés dans les usines, garer une voiture, déplacer les rovers envoyés dans l'espace ou des robots humanoïdes ou encore résoudre les problèmes de pliage de molécules. De nombreuses méthodes ont été proposées pour résoudre la problématique d'une manière générale. On peut se référer à [Latombe 1991; Choset et al. 2005] pour une description plus précise de ces méthodes.

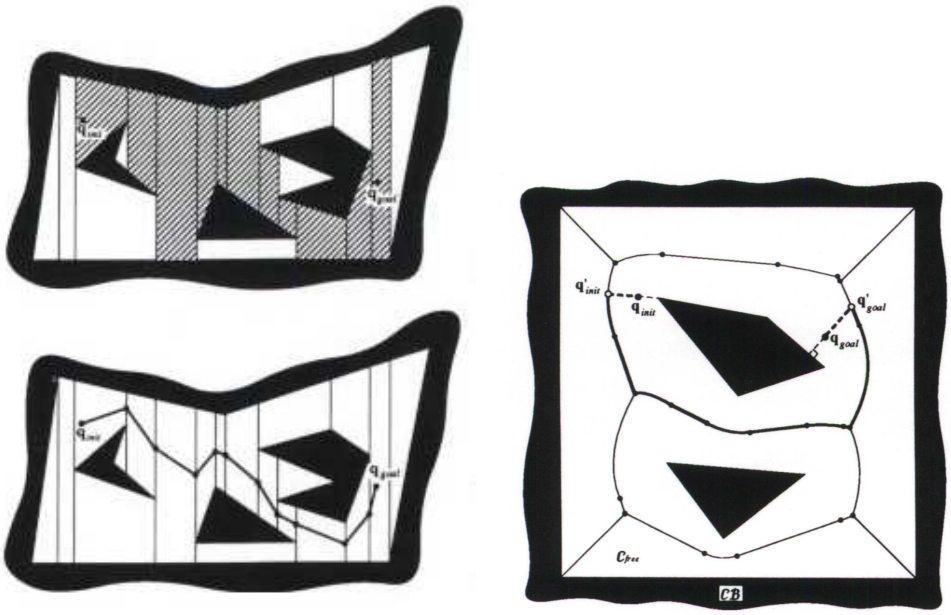


FIG. 2.4 – De [Latombe 1991]. Décomposition cellulaire et Diagramme de Voronoï

Parmi les premières méthodes proposées, on trouve les méthodes dites déterministes. Les

méthodes déterministes peuvent être exactes et sont dites complètes dans ce cas, c'est-à-dire que si une solution existe, elle sera trouvée, sinon la méthode pourra dire qu'il n'y a pas de solution. D'un autre côté, les méthodes déterministes dites approchées ne sont pas complètes mais possèdent certaines propriétés intéressantes, telles que la complétude en résolution (si une solution existe, elle sera trouvée en temps fini). Elles reposent sur le calcul d'une représentation de l'espace des configurations, qui permet de construire un graphe donnant la connexité de l'environnement. Parmi ces méthodes on peut trouver la décomposition en octree, les diagrammes de Voronoï ou encore les graphes de visibilité (figure 2.4). À partir du graphe construit, on peut chercher un chemin partant d'une configuration initiale jusqu'à une configuration finale quelconque, tout en évitant les obstacles. La recherche du chemin solution revient donc à la recherche d'un chemin dans le graphe. Si la configuration initiale et la configuration finale appartiennent à la même composante connexe du graphe, l'existance d'un chemin entre ces deux configurations est garantie.

Mais malgré leurs propriétés, ces algorithmes ne permettent de résoudre en pratique que des problèmes simples avec des espaces des configurations de faible dimension (inférieure à six), car le temps de calcul augmente rapidement avec l'augmentation du nombre de degrés de liberté [Canny 1988]. Pour résoudre des cas pratiques, qui souvent ont un nombre de degrés de liberté supérieur ou égal à six, des algorithmes basés sur l'échantillonnage aléatoire de l'espace des configurations ont été introduits dans les années 1990. Ces méthodes perdent les propriétés des méthodes déterministes (complétude) au profit d'un meilleur temps de calcul. Néanmoins, ces méthodes dites probabilistes garantissent une complétude en probabilité : la probabilité de trouver une solution converge vers 1 quand le temps de calcul tend vers l'infini, si la solution existe.

Ces algorithmes se basent donc sur une méthode d'échantillonnage qui permet de fournir une configuration (alors appelée échantillon) appartenant à l'espace des configurations de l'objet considéré pour construire un graphe représentant CS_{free}. Tout l'intérêt de ces méthodes repose sur le fait qu'elles évitent de construire explicitement l'espace des configurations non-libres. Dans le cas des méthodes déterministes, c'est cette partie qui prend le plus de temps de calcul. En fonction des algorithmes, les graphes construits permettent de capturer la connexité de CS_{free}, car ils peuvent être composés de une ou plusieurs composantes connexes. Une composante connexe d'un graphe est une sous-partie de ce graphe dans laquelle on peut atteindre un nœud à partir de n'importe quel autre nœud, chaque nœud du graphe correspondant à une configuration de CS_{free}.

Parmi les différents algorithmes probabilistes existants, deux méthodes se sont imposées comme méthodes de base dans le domaine de la planification probabiliste. La première a été introduite dans [Kavraki et al. 1996], il s'agit de la méthode « Probabilistic RoadMap » (*PRM*), dont le principe est illustré sur la figure 2.5. L'idée de base de cette méthode est d'échantillonner

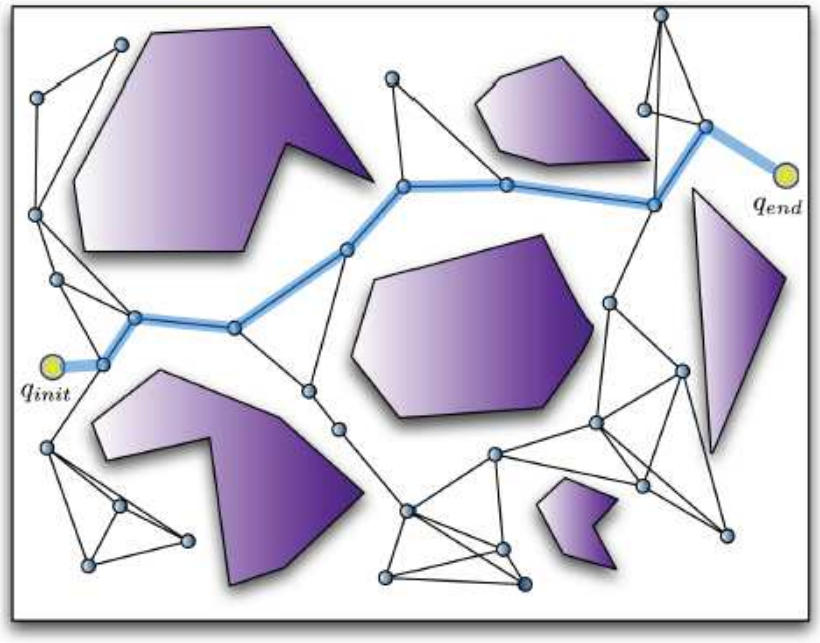


FIG. 2.5 – De [Esteves Jaramillo 2007]. Illustration d'un problème résolu par l'algorithme *PRM*. Le graphe est d'abord construit, puis un chemin est trouvé dans ce graphe pour relier les configurations q_{init} et q_{end}

l'espace des configurations afin de trouver des configurations sans collisions. Chaque configuration échantillonnée est testée pour savoir si elle est libre ou en collision à l'aide d'un détecteur de collisions. Les échantillons sans collisions sont ajoutés à un graphe et la création d'une arête avec les autres nœuds du graphe est tentée. Cette connexion est réalisée grâce à une méthode locale. La méthode locale permet de relier un nœud à un autre avec un chemin libre de collisions. Une méthode locale de base peut être par exemple une simple interpolation linéaire entre les deux nœuds concernés. Lorsqu'une recherche de chemin est demandée, les configurations initiale et finale de la demande sont ajoutées au graphe et des arêtes connectant ces configurations au graphe sont créées. Celui-ci est ensuite parcouru par un algorithme de type $A^\star$ pour trouver le chemin entre les deux configurations. Cette méthode permet de conserver le graphe construit afin de pouvoir effectuer des planifications successives très rapidement dans le même environnement, mais pour des configurations initiale et finale différentes. L'intérêt de ce type de méthode est qu'elles ne calculent pas *CS*, tous les tests de collision étant faits dans W. Dans ce cas, il faut que le graphe contienne suffisamment d'informations pour avoir une couverture représentative de l'espace des configurations libres. Cette étape qui demande un certain temps est généralement effectuée hors-ligne (avant qu'une requête de planification ne soit reçue). Une construction représentative de CS_{free} est très coûteuse, notamment dans le cas où le graphe n'est utilisé qu'une seule fois.

Dans la section suivante est introduite la deuxième méthode de base, qui est une méthode de

diffusion qui permet d'éviter la construction complète du graphe qui représente implicitement CS_{free}.

2.1.1.3 Les arbres de diffusion

Les méthodes de diffusion permettent de construire un graphe rapidement, à usage unique et qui ne nécessite pas forcément de couvrir la totalité de l'espace des configurations pour trouver le chemin solution. Dans cette section nous allons décrire l'une des méthodes de diffusion, les « Rapidly-exploring Random Trees » (*RRT*, introduite dans [LaValle 1998]), qui sera la méthode de base de tous les travaux présentés dans ce mémoire.

L'objectif de l'algorithme *RRT* est de construire un **arbre**, c'est-à-dire un graphe généralement non-orienté et dont les différentes composantes connexes ne contiennent pas de boucles. Chaque nœud du graphe possède donc un seul et unique nœud parent (sauf pour la **racine** de l'arbre, qui est le premier nœud du graphe, en général la configuration initiale) et plusieurs nœuds fils (sauf pour les **feuilles** de l'arbre qui n'en ont pas).

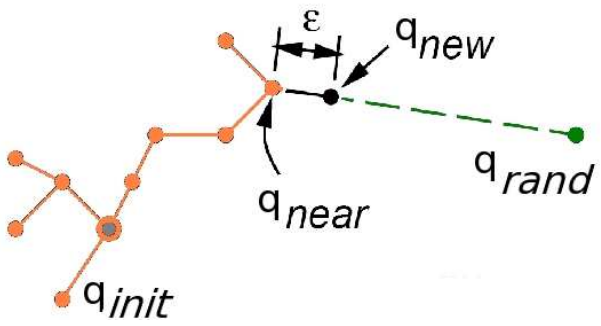


FIG. 2.6 – Illustration du principe d'extension du *RRT*

En partant d'une configuration initiale, l'algorithme *RRT* explore incrémentalement l'espace des configurations pour trouver un chemin atteignant la configuration finale. Ainsi, l'algorithme *RRT* se diffuse de proche en proche pour explorer *CS* et atteindre la configuration finale.

L'algorithme peut se décomposer en cinq étapes (illustrées sur la figure 2.6 et l'algorithme 1) :

- À l'initialisation, la configuration initiale q_{init} est ajoutée au graphe.
- À chaque itération, une nouvelle configuration q_{rand} est échantillonnée (*RANDOM_CONFIG*).
- On recherche ensuite le nœud q_{near} le plus proche de q_{rand} dans l'arbre existant (*NEAREST_NODE*).
- On essaie enfin d'étendre l'arbre à partir de q_{near} dans la direction de q_{rand} d'une longueur ε (*CONNECT*).
- Si l'extension réussit, on ajoute la nouvelle configuration créée q_{new} dans l'arbre et on

recommence à l'étape 2 jusqu'à ce que la configuration finale q_{goal} soit atteinte (une tentative de connexion vers q_{goal}peut être effectuée à une fréquence prédéterminée).

La façon dont l'algorithme *RRT* se diffuse dépend en premier lieu de la méthode d'échantillonnage aléatoire. Une méthode commune est l'échantillonnage uniforme sur l'espace des configurations. Cette méthode permet d'avoir une probabilité équivalente pour chaque configuration de *CS* d'être choisie comme échantillon. Dans beaucoup de méthodes prenant l'algorithme *RRT* comme base, un biais vers la configuration finale est introduit pour diriger la recherche et ne pas effectuer uniquement une exploration aléatoire. Dans l'algorithme 1, ainsi que dans les autres algorithmes de ce document, N est un nombre d'itérations fixe décidé empiriquement qui représente la limite autorisée pour résoudre le problème (les algorithmes de diffusion aléatoire ne donnant aucune garantie quant au temps de résolution).

Algorithm 1 L'algorithme RRT-CONNECT

$T(q_{init})$
for $i = 0$ to N **do**
  $q_{rand} \leftarrow$ RANDOM_CONFIG()
  $q_{near} \leftarrow$ NEAREST_NODE(q_{rand}, T)
  if CONNECT(T, q_{rand},q_{near},q_{new}) **then**
    Add_Vertex(T, q_{new})
    Add_Edge(T, q_{near},q_{new})
  end if
end for

Plusieurs variantes de cette méthode ont ensuite été développées [LaValle 2006]. Parmi celles-ci, on peut citer l'algorithme *RRT* bi-directionnel [LaValle and Kuffner 1999] dont le principe est de construire deux arbres de diffusion, l'un ayant sa racine à la configuration initiale et l'autre à la configuration finale. La résolution du problème est ainsi plus rapide en probabilité car les deux arbres s'étendent l'un vers l'autre. L'algorithme *RRT* « connect » [Kuffner and LaValle 2000] permet une exploration plus rapide de *CS* car l'extension de q_{near} vers q_{rand} est maximisée : on étend l'arbre jusqu'à q_{rand} ou jusqu'à rencontrer un obstacle. Ainsi l'espace libre est exploré très rapidement jusqu'aux obstacles les plus proches. Ces méthodes probabilistes ont néanmoins des limites : une couverture suffisante de *CS* pour trouver la solution n'est pas toujours garantie, notamment dans le cas où il y a des passages étroits dans l'environnement. De plus, ces méthodes ne permettent pas de détecter l'absence de solution et d'arrêter l'échantillonnage de *CS* s'il n'y en a pas.

2.2 Travaux précédents

2.2.1 Les arbres locaux

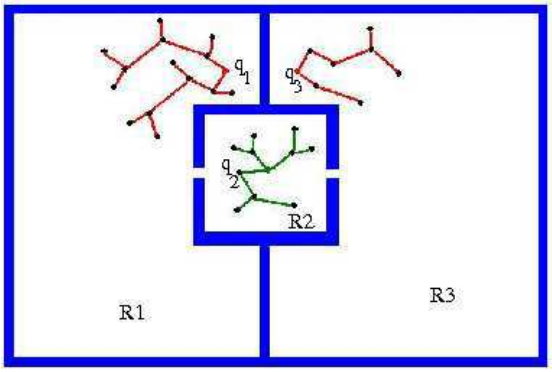


FIG. 2.7 – Un exemple d'arbres locaux. La méthode a de fortes probabilités de générer des nœuds dans les trois régions R_i. Le résultat sera une diffusion de trois arbres dans ces trois régions. La probabilité de connecter deux arbres sera plus grande s'il y a un arbre de plus dans la région R_2 que s'il n'y a que les deux arbres initiaux dans les régions R_1 and R_3.

Dans [Strandberg 2004], l'auteur introduit une méthode basée sur les arbres locaux, que nous noterons *LTRRT*, qui sont des composantes connexes supplémentaires ajoutées au graphe et dont la diffusion ne se fait pas à partir des configurations initiale et finale. Ces arbres locaux permettent d'augmenter le nombre de nœuds du graphe créés dans les régions de l'environnement difficilement accessibles depuis les seules composantes diffusées à partir des configurations initiale et finale. En diffusant des arbres locaux qui partent de l'intérieur des zones, la connexion avec le reste du graphe à travers les passages étroits (définissant donc une région difficile d'accès) est facilitée.

L'idée de base de la méthode des arbres locaux est illustrée sur la figure 2.7. Sur cette figure, on peut voir que si un arbre local $T(q_2)$ (dont la racine de diffusion est la configuration q_2) est diffusé au milieu de la région $R2$, la probabilité de connecter les arbres $T(q_1)$ et $T(q_2)$ ou les arbres $T(q_2)$ et $T(q_3)$ sera supérieure à la probabilité de connecter directement $T(q_1)$ et $T(q_3)$

Cette approche est une méthode héritant à la fois de l'algorithme *RRT* basique, auquel on reviendrait si on autorisait au maximum deux arbres de diffusion et de l'algorithme *PRM*, qui autoriserait l'ajout dans le graphe de chaque échantillon en tant que racine de diffusion, mais en interdisant toute diffusion.

Les performances de cette méthode en temps de calcul et en place mémoire dépendent fortement de la façon dont sont traités les trois problèmes principaux qu'elle pose : quand peut-on créer un arbre local ? Quand peut-on autoriser la diffusion de ces arbres ? Quand doit-on arrêter leur diffusion ?

Dans [Strandberg 2004], l'auteur répond à ces questions en utilisant des heuristiques. Le nombre d'arbres locaux créés est limité. Si la limite est atteinte, alors l'extension d'un arbre local sera autorisé avec une probabilité P_{grow}. La fusion de deux arbres ne sera testée que si la boîte englobante d'un arbre s'est agrandie. L'approche présente un intérêt certain pour la couverture de l'espace des configurations, mais lorsque le nombre d'arbres de diffusion augmente, le temps de calcul augmente également rapidement car le nombre de nœuds de chaque arbre local, ainsi que le nombre de connections possibles entre ces arbres, augmentent aussi. Un moyen d'améliorer le comportement de l'algorithme est de limiter le nombre de nœuds ajoutés au graphe et de limiter le nombre d'arbres locaux qu'il est possible de créer.

Cette idée est à la base d'une variante originale de la méthode *PRM* qui utilise la notion de visibilité entre les nœuds d'un graphe pour créer un graphe appelé graphe de visibilité.

2.2.2 Graphes probabilistes de visibilité

Cette méthode est décrite dans [Siméon et al. 2000] sous le nom de Visibility-PRM (*VISPRM*). Les auteurs définissent deux types de nœuds différents pour le graphe lors de la construction :

- Les nœuds « gardiens » qui permettent de couvrir le plus largement possible l'espace des configurations libres.
- Les nœuds « connecteurs » qui permettent de réduire le nombre de composantes connexes du graphe en les connectant entre elles.

Pour construire ce graphe, l'algorithme n'ajoute que les configurations qui permettent soit d'augmenter la visibilité sur l'espace des configurations libres (les nœuds gardiens, qui ne peuvent être reliés les uns aux autres directement) soit de relier deux composantes connexes du graphe (les nœuds connecteurs). Dans CS, la notion de **visibilité**, ou domaine de visibilité d'une configuration q, peut se définir comme étant l'ensemble des configurations q_x pouvant être directement connectées à q par une méthode locale. Les connexions entre q et les q_x appartiennent à CS_{free}. Le graphe produit par cette méthode contient donc un minimum de nœuds tout en couvrant un maximum de l'espace des configurations libres, à la différence de l'algorithme *PRM* classique, comme on peut le voir sur la figure 2.8. Une des propriétés intéressantes est de contrôler la fin de la construction du graphe en estimant la couverture de cet espace.

L'inconvénient de cette méthode est qu'elle rend difficile la connexion entre deux composantes dont le domaine de visibilité commun est très faible (ce qui est souvent lié à un passage étroit séparant les deux composantes), comme nous le verrons dans le paragraphe suivant.

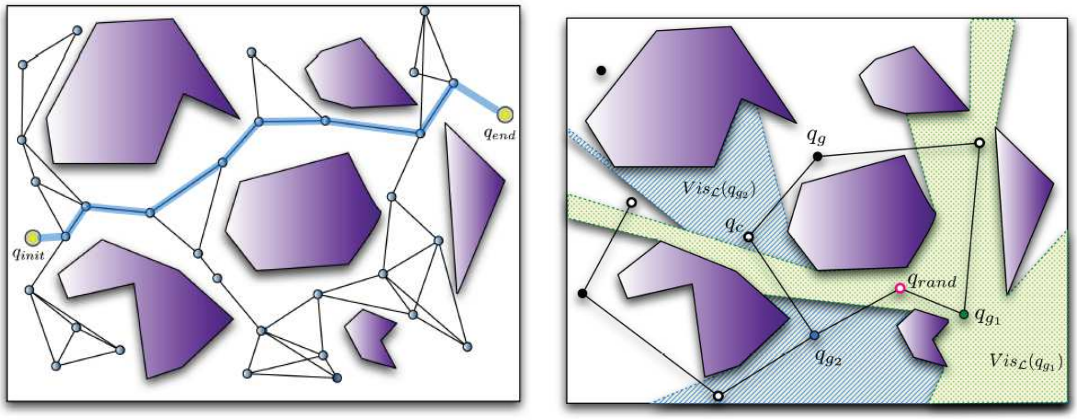


FIG. 2.8 – De [Esteves Jaramillo 2007]. Comparaison des méthodes *PRM* (à gauche) et *VISPRM* (à droite). Pour l'algorithme *VISPRM*, on peut apercevoir en vert et en bleu les zones de visibilité des nœuds q_{g_1} et q_{g_2}.

2.3 Le planificateur Arbres Locaux de visibilité

L'idée principale de la méthode des arbres locaux de visibilité (*VISLT*) est d'utiliser à la fois les avantages des graphes de visibilité et des arbres locaux pour résoudre les inconvénients de chacune des deux méthodes.

2.3.1 Principe

Comme cela à été dit précédemment, la méthode des arbres locaux nécessite la résolution de trois principaux problèmes pour être réellement efficace :

- Où et quand initier la diffusion d'un arbre local ?
- Quand peut-on autoriser l'arbre local à s'étendre ?
- Quand doit-on arrêter la croissance de l'arbre local ?

Ces trois problèmes peuvent être résolus tous à la fois en utilisant la méthode *VISPRM*. Les arbres locaux permettent l'exploration des zones qui ne sont pas directement accessibles depuis les arbres partant de la configuration initiale et de la configuration finale et des zones qui seront de toute façon difficilement accessibles par la suite (à cause de la présence de passages étroits menant à ces zones, par exemple). La présence d'arbres locaux se diffusant directement depuis l'intérieur de ces zones permet de les connecter au reste du graphe beaucoup plus facilement, en probabilité.

Les arbres locaux doivent donc démarrer leur croissance à l'intérieur de ces zones. Mais comment définir ce point de départ ? La notion de nœuds gardiens introduite dans la méthode des graphes de visibilité répond parfaitement à ce problème. En effet, ces nœuds sont créés lorsque la configuration échantillonnée correspondante ne peut pas être liée directement aux

autres nœuds du graphe, définissant donc une zone relativement isolée autour de cette position. Les nœuds gardiens des graphes de visibilité permettent donc de répondre au premier problème des arbres locaux.

Ces arbres locaux créés dans des zones isolées doivent ensuite pouvoir être liés au reste du graphe. Avec la méthode des graphes de visibilité, les nœuds connecteurs ne sont créés que s'ils peuvent lier deux composantes connexes du graphe. Mais dans le cas de passages étroits, la probabilité que cette liaison se crée est faible car elle dépend du domaine de visibilité et ces passages réduisent considérablement le domaine de visibilité commun aux composantes présentes de chaque côté. D'un autre côté, plus un nœud est proche du passage en question, plus son domaine de visibilité est étendu, ce qui augmente donc les probabilités de connexion comme on peut le voir en comparant les figures 2.9 et 2.10.

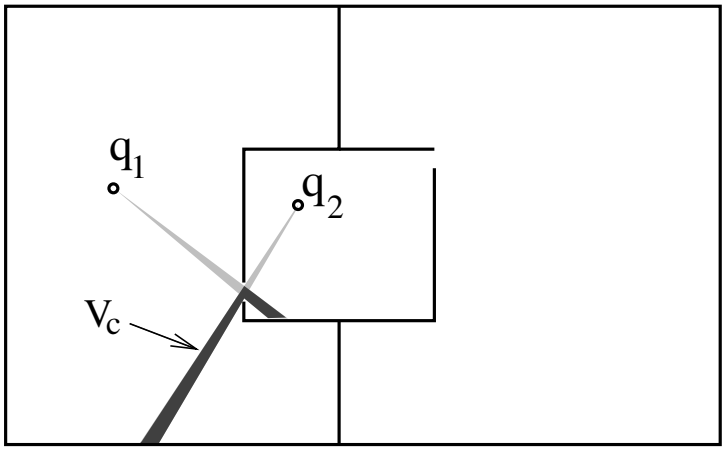


FIG. 2.9 – Un exemple de la difficulté de créer un nœud de connexion avec la méthode Visibility PRM. Un nœud de connexion ne peut être créé que si un échantillon est tiré dans la région V_C, région de visibilité commune de q_1 et q_2. La probabilité d'avoir une configuration échantillonnée dans la région V_C (en gris foncé) est faible.

Partant de ce constat, nous avons introduit la notion de nœuds « éclaireurs » dans la méthode des arbres locaux de visibilité. Ces nœuds sont créés lorsqu'ils permettent de se rapprocher des passages étroits. Comme pour toute méthode probabiliste, *CS* n'est pas construit explicitement aussi il est impossible de savoir a priori où se situent lesdits passages étroits et a fortiori de s'en rapprocher explicitement avec ces nœuds éclaireurs. Nous partons donc de l'observation suivante : si un arbre local n'a pas encore été connecté au reste du graphe (ou à un autre arbre local), c'est que son domaine de visibilité n'est pas assez grand pour permettre cette connexion et qu'il est donc relativement éloigné des passages étroits entourant sa zone. Un nœud s'éloignant de l'arbre (c'est-à-dire plus éloigné de la racine de l'arbre que son nœud parent) s'approchera des limites de la zone et donc potentiellement des passages étroits, augmentant

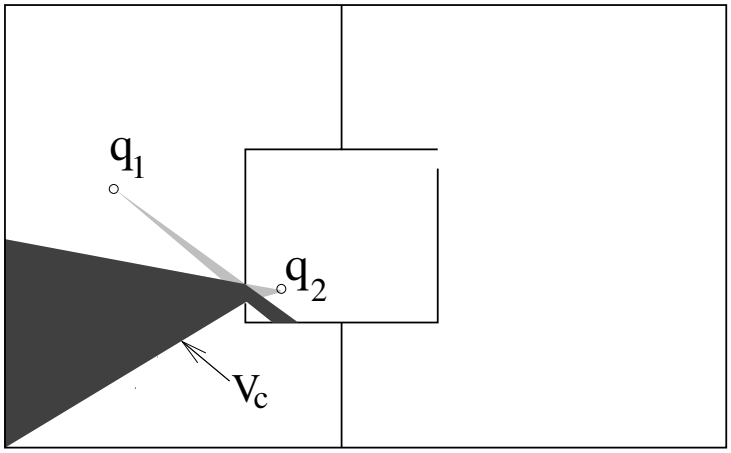


FIG. 2.10 – Comparer le domaine d'échantillonnage V_C sur cette figure à celui sur la figure 2.9. Le domaine de visibilité commun à q_1 et q_2 est plus grand car q_2 est plus proche du passage étroit. La probabilité d'avoir un échantillon dans V_c est donc plus grande.

ainsi le domaine de visibilité de l'arbre. C'est le rôle de nœuds éclaireurs. L'introduction des nœuds éclaireurs permet donc de répondre au deuxième problème des arbres locaux, à savoir la croissance des arbres créés.

Mais on ne peut faire croître tous les arbres locaux indéfiniment car on augmente le temps de résolution du problème à cause de tous les tests de collision. Dans le pire des cas, un arbre local situé dans une zone close (réellement inaccessible depuis les configurations de départ et de fin du problème) consommera inutilement un temps important à faire des tests de collisions ne permettant pas de gagner de l'information supplémentaire pour la résolution. Il faut donc définir un critère d'arrêt de la croissance d'un arbre pour éviter ce problème. Dans les graphes de visibilité, la croissance du graphe est arrêtée en fixant un seuil de temps ou de nombre de nœuds, ou bien dès que le nombre de nœuds créés n'augmente plus alors que l'algorithme continue d'échantillonner des configurations. Dans ce cas, on peut dire que l'espace des configurations libres est presque totalement couvert par le graphe puisque aucun nœud gardien n'est créé. Dans la méthode des arbres locaux de visibilité, nous avons un comportement du même type pour chaque arbre. En fonction du seuil de densité des nœuds acceptés, l'échantillonnage cessera de produire de nouvelles configurations libres de collision pour un arbre donné au bout d'un certain temps. En effet, lorsque l'arbre local a atteint les limites de sa zone, aucun nœud éclaireur ne peut plus être ajouté. Cette dernière propriété permet de résoudre le dernier problème des arbres locaux concernant l'arrêt de la croissance des arbres.

La méthode des arbres locaux de visibilité résultant de l'intégration de ces trois points est une méthode basée sur la méthode *RRT* combinant les deux stratégies, chacune résolvant les problèmes de l'autre.

2.3.2 Implémentation

Algorithm 2 L'algorithme Visibility Local Trees

$F \leftarrow T_{q_{init}}, T_{q_{goal}}$
$Guardians \leftarrow \emptyset$
$nbPossibleConnection = 0$
for $j = 0$ to N **do**
  $q_{rand} \leftarrow$ RANDOM_CONFIG()
  for All Trees T_i in F **do**
    $q_{near_i} \leftarrow$ NEAREST_NEIGHBOR(q_{rand}, T_i)
    if CAN_CONNECT(T_i, q_{rand},q_{near_i}) **then**
      $nbPossibleConnection++$
    end if
  end for
  if $nbPossibleConnection = 0$ **then**
    $F \leftarrow q_{rand}$
    $Guardians \leftarrow q_{rand}$
    $T_{new} =$ ADD_NEW_TREE(q_{rand})
    $F \leftarrow T_{new} \cup F$
  else
    if $nbPossibleConnection = 1$ **then**
      $T_n \leftarrow$ CONNECTABLE_TREE(q_{rand})
      $q_{near_n} \leftarrow$ NEAREST_NEIGHBOR(q_{rand},T_n)
      if DIST_ROOT(q_{rand},T_n)> DIST_ROOT(q_{near_n},T_n) **then**
        Add_Vertex(T_n, q_{rand})
        Add_Edge(T_n, q_{near_n},q_{rand})
      end if
    end if
  else
    for All Connectable Trees T_i in F **do**
      $q_{near_i} \leftarrow$ NEAREST_NEIGHBOR(q_{rand}, T_i)
      Add_Vertex(T_i, q_{rand})
      Add_Edge(T_i, q_{near_i},q_{rand})
    end for
  end if
end for

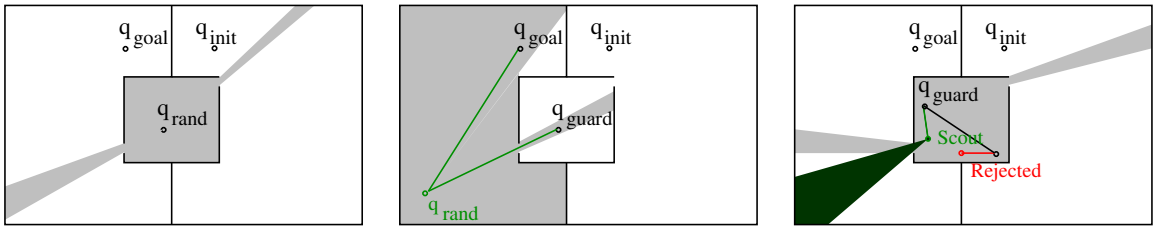
L'algorithme des arbres locaux de visibilité décrit dans l'algorithme 2 est divisé en trois parties, chacune correspondant à une des étapes suivantes :

Dans un premier temps, l'étape d'échantillonnage, qui est une étape commune dans les méthodes probabilistes telles que les méthodes *RRT* ou *PRM*, nous permet d'obtenir un nouvel échantillon aléatoirement q_{rand} libre de collisions. Pour la méthode des arbres locaux de visibilité, nous avons choisi d'utiliser un échantillonnage uniforme sur l'espace

des configurations. L'espace des configurations est donc explicitement borné afin de pouvoir effectuer un tirage aléatoire uniforme.

Ensuite, pour chaque arbre local T_i (aussi appelé composante connexe) dans le graphe, nous essayons de savoir si la connexion à la configuration obtenue à l'étape précédente est possible, sans effectuer réellement la connexion dans le graphe. La connexion n'est pas effectuée directement dès le début car le traitement de la configuration échantillonnée variera en fonction du nombre d'arbres locaux qui pourront y être connectés. Il est donc nécessaire de garder en mémoire le nombre d'arbres pouvant être connectés (*nbPossibleConnection*).

Au lieu de tester cette connexion pour chaque nœud de chaque composante connexe du graphe, nous avons choisi de n'effectuer ce test que sur le nœud le plus proche de la configuration échantillonnée pour chaque composante connexe. Cela permet d'accélérer cette phase préalable à l'extension réelle du graphe. Ce choix influe légèrement sur la vitesse de connexion des arbres locaux au reste du graphe et augmente donc temporairement le nombre de composantes connexes dans ce graphe.



(a) (b) (c)

FIG. 2.11 – Illustration de la création des trois différents types de nœuds : (a) Un nœud gardien (q_{rand}, au milieu), (b) Un nœud connecteur (q_{rand}, en vert à gauche) et (c) Un nœud éclaireur (en vert au centre). Le nœud rouge est plus proche de la racine que son nœud parent, il est donc rejeté.

En fonction du nombre de connections possibles calculé précédemment, trois cas de figure sont pris en compte pour décider de l'ajout ou non de la configuration échantillonnée :

- Si aucune connexion n'est possible entre q_{rand} et une des composantes connexes du graphe, q_{rand} est ajoutée en tant que nœud gardien dans le graphe F et devient la racine d'un nouvel arbre local (Figure 2.11-(a)).
- S'il n'y a qu'une seule connexion possible, la distance de q_{rand} par rapport à la racine de l'arbre T_n est calculée. Cette distance est ensuite comparée à la distance à la racine du nœud du graphe concerné par la connexion (qui sera donc le nœud parent en cas de connexion effective). Si le nœud potentiellement parent est le plus proche de la racine, c'est-à-dire que la distance (DIST_ROOT) de q_{near_n} à la racine est plus petite que la distance de q_{rand} à la racine, alors q_{rand} est connectée et ajoutée au graphe en tant que nœud éclaireur. La distance calculée pour ce test est mémorisée dans la structure du

nœud, ce qui permet de gagner du temps lors des comparaisons suivantes. Dans notre implémentation, nous avons ajouté un coefficient c_{dist} imposant un certain ratio entre les deux distances, afin d'augmenter la vitesse d'extension de l'arbre. Sur la figure 2.11-(c), on peut voir que l'un des échantillons (en rouge) a été rejeté car il est plus près de la racine de la composante connexe que son nœud parent. Par contre, le nœud éclaireur (en vert) a été ajouté car son nœud parent est la racine elle-même (ce nœud est plus proche de la racine que n'importe quel autre nœud de la composante connexe).

– S'il y a deux connections ou plus de possible, la configuration q_{rand} est ajoutée en tant que nœud connecteur et les connections entre les différentes composantes connexes sont effectuées. Ces composantes sont donc fusionnées en une seule. Une nouvelle racine est choisie pour celle-ci (parmi les racines des composantes fusionnées) et les distances de chaque nœud à leur nouvelle racine sont calculées. Le voisin le plus proche dans chaque composante q_{near_i} est utilisé ici pour les connexions. Celui-ci n'est calculé qu'une seule fois (NEAREST_NEIGHBOR) lors de l'étape préalable à la connexion. Sur la figure 2.11-(b), le nœud connecteur q_{rand} (en vert) se situe dans le domaine de visibilité commun de deux composantes connexes, dont les racines sont q_{guard} et q_{goal}.

2.3.3 Expérimentations

Cet algorithme a été implémenté en C++ sur la base de la plateforme HPP (Humanoïd Path Planner) développée au LAAS et basée elle-même sur KineoWorks. KineoWorks est un kit de développement logiciel dédié à la planification de mouvement et développé par la société KineoCAM.

Les expérimentations ont été faites sur un PC Dual Core, 2,1 GHz avec 2Go de RAM. Toutes les valeurs reportées dans les tableaux sont des moyennes calculées sur 100 essais. Dans nos tests, nous avons comparé la méthode du *VISLT* avec les deux méthodes sur lesquelles elle se base, à savoir les méthodes *LTRRT* et *VISPRM*, ainsi qu'avec la méthode *RRT* de base. Les fonctions de base de l'algorithme *RRT* (méthode d'échantillonnage, détection de collision, calcul du plus proche voisin) sont les mêmes pour les différentes méthodes comparées et pour chaque test. La méthode d'échantillonnage est uniforme sur l'espace des configurations (qui est explicitement borné). Un échantillonnage uniforme permet de choisir avec une probabilité équivalente toutes les configurations de *CS* à tout moment lors de l'algorithme. Un biais vers le but est ajouté. La détection de collisions est effectuée par le moteur de KineoWorks, nommé KCD. La recherche du plus proche voisin se fait de manière simple, en parcourant tout les nœuds de la composante connexe concernée lors de l'itération. Dans le cas des nœuds éclaireurs pour les arbres locaux de visibilité, un coefficient est appliqué aux distances comparées afin de pourvoir influencer plus ou moins l'ajout de nœuds éclaireurs. Pour ajouter le nouveau nœud, sa distance à la racine doit être supérieure d'au moins 10% à celle du nœud parent. Ce coefficient,

nommé c_{dist}, est donc fixé à 1,1.

Dans un premier temps, nous avons testé et comparé notre méthode sur plusieurs exemples : des environnements simples avec un système possédant trois degrés de liberté permettent de mieux concevoir l'intérêt de la méthode, ses avantages et ses performances par rapport aux méthodes comparées. Ces exemples permettent aussi de se rendre compte de l'importance des passages étroits dans un problème de planification de mouvement, par rapport à la dimension de l'espace des configurations. Des exemples plus complexes avec des espaces des configurations de dimensions supérieures sont ensuite utilisés pour démontrer les avantages présentés lors de la description de la méthode.

2.3.4 Résultats

2.3.4.1 Environnements simples à trois et cinq pièces

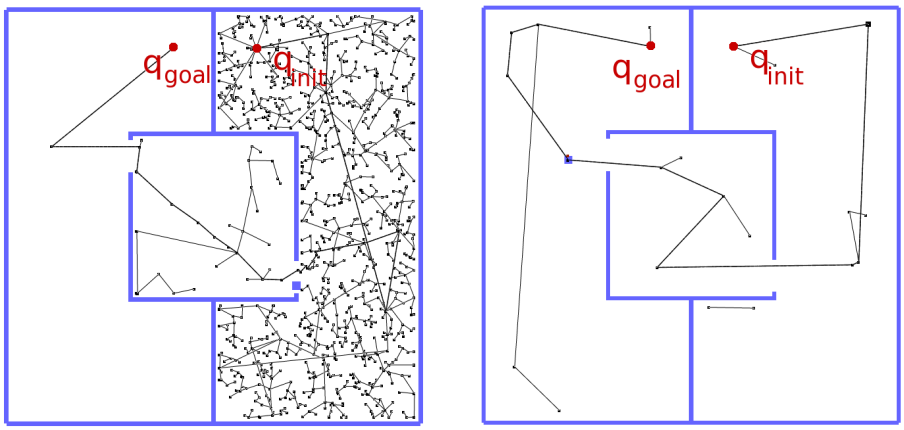


FIG. 2.12 – Les graphes pour l'algorithme *RRT* (gauche) et notre méthode (droite) dans un espace des configurations de dimension 2, avec deux passages étroits.

Dans ces premiers tests, nous avons utilisé des environnements composés de plusieurs pièces séparées par un nombre variable de passages étroits, avec une taille variable.

Les premiers tests ont été effectués sur des environnements plans avec un robot polygonal possédant 2 *ddl* en translation avec un espace des configurations de dimension 2 et un espace de travail de dimension 2. L'environnement est borné et comporte trois pièces séparées par de petites ouvertures plus ou moins étroites par rapport au robot (figure 2.12). Le robot est un carré de taille 10x10. L'environnement est de taille 500x500. La largeur des ouvertures est définie par le paramètre K dans les tableaux de résultats :

$$K = \frac{\texttt{largeur du passage}}{\texttt{largeur du robot}} \tag{2.1}$$

On peut voir sur la figure 2.12 que le nombre de nœuds du graphe de la méthode des arbres locaux de visibilité est très inférieur à celui du graphe de l'algorithme *RRT* basique, ce qui est confirmé dans le tableau 2.1. Ce tableau présente les résultats pour les quatre méthodes comparées, avec quatre tailles d'ouverture différentes (de la plus grande à la plus petite). Les résultats présentés permettent de voir que les arbres locaux de visibilité sont plus efficaces que les méthodes comparées.

TAB. 2.1 – Résultats pour un environnement à trois pièces avec deux passages étroits et un espace des configurations de dimension 2.

Cas	Algorithme	Itérations	Nœuds	Temps
(K=3)	RRT	1057	474	34 s
	LTRRT	164	115	12 s
	VISPRM	72	8	6 s
	VISLT	90	51	4 s
(K=2.5)	RRT	1569	690	52 s
	LTRRT	356	221	25 s
	VISPRM	206	8	16 s
	VISLT	130	69	5 s
(K=2)	RRT	2552	1116	89 s
	LTRRT	499	291	33 s
	VISPRM	544	9	60 s
	VISLT	157	80	7 s
(K=1.5)	RRT	3874	1647	134 s
	LTRRT	905	487	59 s
	VISPRM	1523	9	128 s
	VISLT	295	141	12 s

Dans un deuxième temps, nous avons effectué des tests sur un environnement avec un espace des configurations de dimension trois (X,Y,θ), comme illustré à la figure 2.13. Le robot à été modifié pour rendre compte de l'importance du paramètre angulaire θ. Les mêmes tailles d'ouverture ont été utilisées pour ces tests.

De la même manière que pour les tests précédents, on voit que la méthode *VISLT* est plus performante en termes de temps et de nombre d'itérations. Le tableau 2.2 présente les résultats pour ces tests.

Pour les tests suivants, un environnement avec cinq pièces séparées chacune par un passage étroit de taille variable est utilisé. Cet environnement est représenté sur la figure 2.14 pour un espace des configurations de dimension 2 (X,Y) et sur la figure 2.15 pour un espace des configurations de dimension 3 (X,Y,θ). Les tableaux 2.3 et 2.4 présentent les résultats des tests

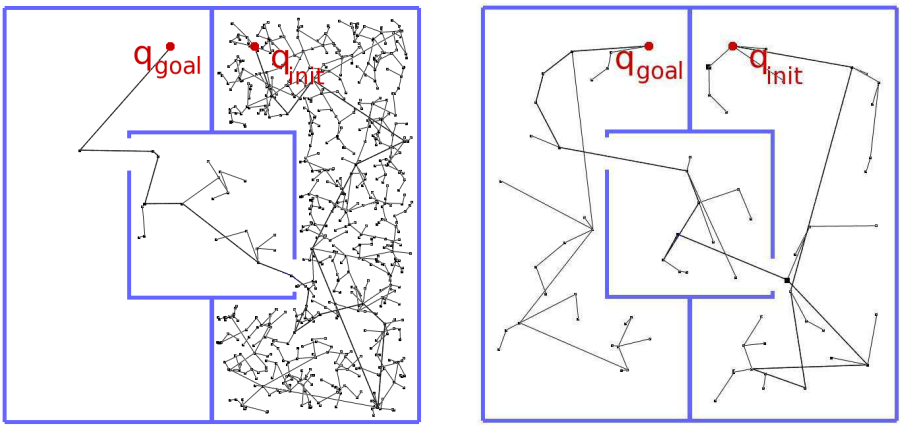


FIG. 2.13 – Les graphes pour un algorithme *RRT* (gauche) et notre méthode (droite) dans un espace des configurations de dimension 3, avec deux passages étroits.

TAB. 2.2 – Résultats pour un environnement à trois pièces avec deux passages étroits et un espace des configurations de dimension 3.

Cas	Algorithme	Itérations	Nœuds	Temps
(K=3)	RRT	4111	1841	131 s
	LTRRT	687	370	43 s
	VISPRM	399	10	31 s
	VISLT	281	118	11 s
(K=2.5)	RRT	6244	2475	194 s
	LTRRT	1360	683	84 s
	VISPRM	606	11	48 s
	VISLT	436	176	16 s
(K=2)	RRT	14650	5498	552 s
	LTRRT	2471	1159	151 s
	VISPRM	1052	12	90 s
	VISLT	780	303	27 s
(K=1.5)	RRT	22470	8606	897 s
	LTRRT	4576	2067	286 s
	VISPRM	2484	14	223 s
	VISLT	1876	700	67 s

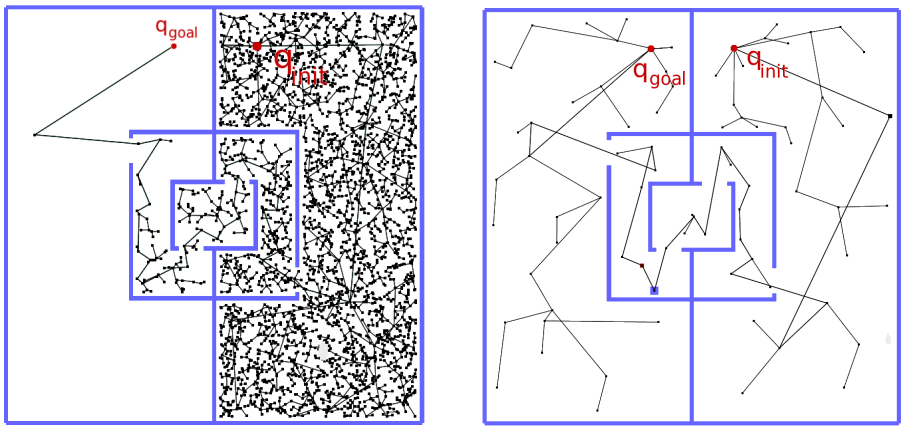


FIG. 2.14 – Les graphes pour l'algorithme *RRT* (gauche) et notre méthode (droite) dans un espace des configurations de dimension 2, avec quatre passages étroits.

pour des espaces des configurations de dimension deux et trois respectivement.

TAB. 2.3 – Résultats pour un environnement à cinq pièces avec quatre passages étroits et un espace des configurations de dimension deux.

Cas	Algorithme	Itérations	Nœuds	Temps
(K=3)	RRT	10289	4412	335 s
	LTRRT	1303	632	83 s
	VISPRM	657	19	106 s
	VISLT	499	242	25 s
(K=2.5)	RRT	12967	5508	455 s
	LTRRT	2301	1077	149 s
	VISPRM	1107	21	195 s
	VISLT	693	322	34 s
(K=2)	RRT	23860	9964	922 s
	LTRRT	4183	1756	274 s
	VISPRM	3516	25	588 s
	VISLT	1223	555	62 s
(K=1.5)	RRT	61198	25198	3381 s
	LTRRT	16510	6349	1156 s
	VISPRM	8278	94	1404 s
	VISLT	3825	1663	217 s

Sur ces différents cas, on peut voir que l'importance de la dimension de l'espace des configurations est moindre que celle du nombre de passages étroits. En effet, en comparant les tableaux 2.1 et 2.3 on voit que la différence est notablement plus importante que celle entre les tableaux 2.1 et 2.2. De même pour les tableaux 2.2 et 2.4 où la différence entre les résultats

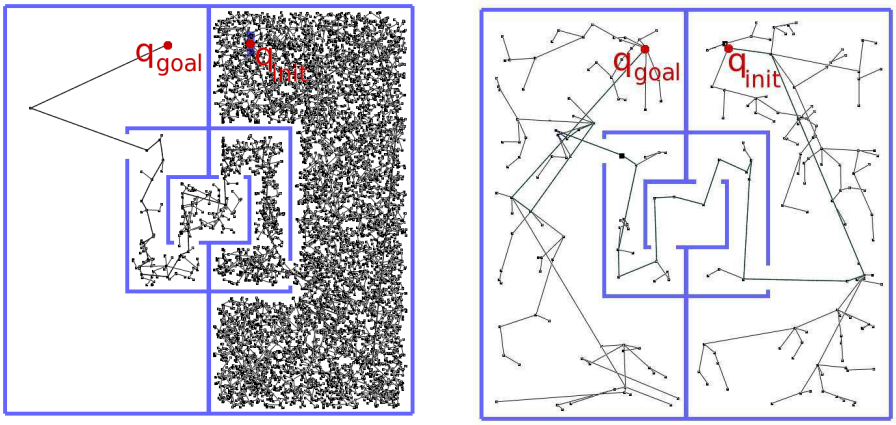


FIG. 2.15 – Les graphes pour l'algorithme *RRT* (gauche) et notre méthode (droite) dans un espace des configurations de dimension 3, avec quatre passages étroits.

TAB. 2.4 – Résultats pour un environnement à cinq pièces avec quatre passages étroits et un espace des configurations de dimension trois.

Cas	Algorithme	Itérations	Nœuds	Temps
(K=3)	RRT	20426	8029	925 s
	LTRRT	3053	1297	292 s
	VISPRM	2539	38	368 s
	VISLT	1396	196	88 s
(K=2.5)	RRT	35246	13631	1916 s
	LTRRT	6105	2405	460 s
	VISPRM	11081	106	1728 s
	VISLT	2543	284	152 s
(K=2)	RRT	77590	29418	5182 s
	LTRRT	16931	6154	1433 s
	VISPRM	46215	316	7380 s
	VISLT	7639	557	410 s
(K=1.5)	RRT	190310	70098	18741 s
	LTRRT	74787	23490	7760 s
	VISPRM	110220	704	18429 s
	VISLT	33149	1363	2027 s

est moindre que celle entre les résultats des tableaux 2.1 et 2.2. Nous ferons une analyse de ces résultats dans le paragraphe 2.3.4.4

2.3.4.2 Le labyrinthe

Dans les tests suivants, nous considérons un espace de travail de dimension 3 et un espace des configurations de dimension 3 (X,Y,Z) pour un robot polygonal avec 3 *ddl* en translation. Le labyrinthe est formé de pièces quasi-planes (les déplacements ne sont contraints que sur une dimension) reliées par des couloirs étroits. L'environnement est représenté sur la figure 2.16, avec le point de départ en haut à gauche et à l'arrière tandis que l'arrivée se situe en bas à droite et à l'avant de la scène. C'est un cas très complexe dans lequel l'algorithme *RRT* n'a pas réussi à trouver de solution après plusieurs heures de calculs. Le tableau 2.5 montre les résultats obtenus pour les 4 algorithmes. Pour des raisons de visualisation nous n'avons pas représenté les murs qui englobent le labyrinthe et qui obligent le robot à se déplacer à l'intérieur.

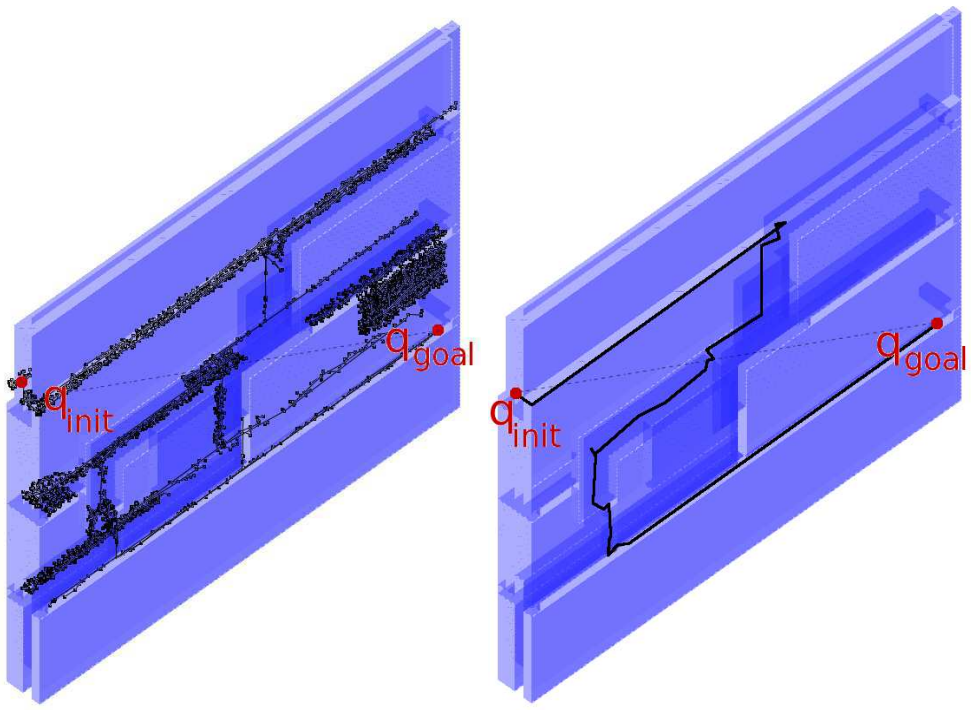


FIG. 2.16 – Un labyrinthe en 3D avec le graphe final de la méthode *VISTL* des arbres locaux de visibilité et la solution (à droite).

TAB. 2.5 – Cas du labyrinthe en trois dimensions.

Labyrinthe 3D	Algorithme	Itérations	Nœuds	Temps
	RRT	259122	2068	>7200 s
	LTRRT	184664	2415	>7200 s
	VISPRM	586276	49	6215 s
	VISLT	347543	567	1613 s

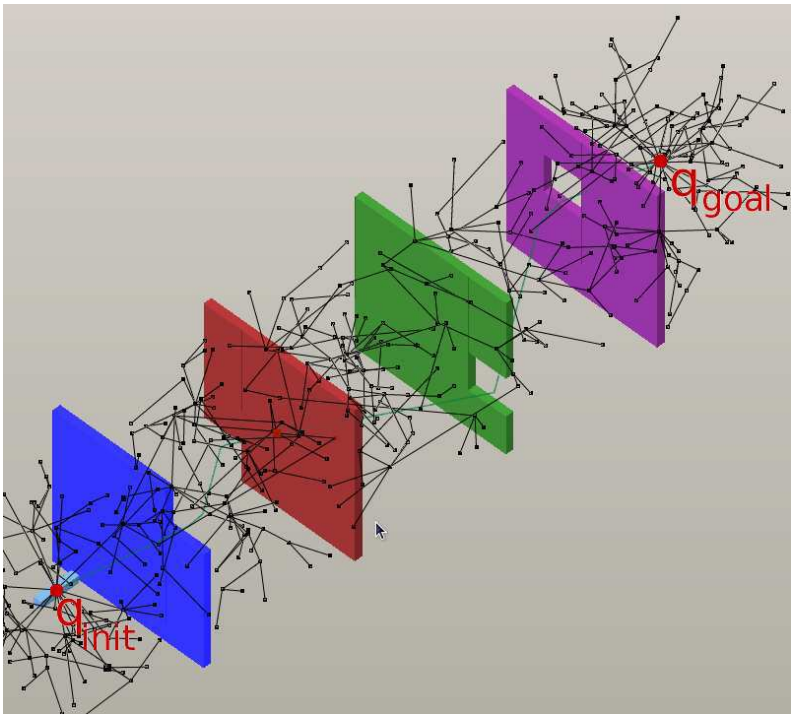


FIG. 2.17 – Un environnement avec quatre passages étroits et un espace des configurations de dimension 6.

TAB. 2.6 – Résultats pour un environnement avec des passages de 25x25 et un espace des configurations de dimension 6.

2 Murs 5p0	Algorithme	Itérations	Nœuds	Temps
	LTRRT	1009	531	92 s
	VISPRM	2585	6	143 s
	VISLT	427	154	28 s
4 Murs 5p0	LTRRT	6166	2824	557 s
	VISPRM	4344	12	511 s
	VISLT	2745	553	166 s
6 Murs 5p0	LTRRT	9925	3833	948 s
	VISPRM	8076	44	1183 s
	VISLT	4108	742	273 s
8 Murs 5p0	LTRRT	18629	7186	1942 s
	VISPRM	12031	81	2328 s
	VISLT	7341	1114	601 s

2.3.4.3 Les murs

Dans ces tests, dont l'espace de travail est illustré sur la figure 2.17, l'espace des configurations est de dimension 6 (X,Y,Z,ϕ,ρ,θ) et le robot est polygonal avec 3 *ddl* en translation et 3 *ddl* en rotation. Le but est de traverser chaque mur en empruntant le passage étroit (trou) qui y est placé différemment sur chacun. Les dimensions de chaque mur sont de 100x100x5 et pour chaque passage de 25x25 à l'intérieur du mur, pour la première série de tests (nommée 5p0 dans les tableaux et figures). Pour la seconde série de tests, les passages ont une taille de 20x20. Les dimensions du robot sont de 5x5x25. Les algorithmes ont été testés sur quatre environnements qui diffèrent par leur nombre de murs (et donc de passages étroits à traverser) : 2, 4, 6 et 8 murs. On peut voir sur les tableaux 2.6 et 2.7 que la méthode *VISLT* a résolu les problèmes en moins de temps et d'itérations que les autres méthodes. Le nombre de nœuds dans le graphe final (après la résolution du problème) est même inférieur pour les arbres locaux de visibilité que pour les arbres locaux basique. Pour ces tests, l'algorithme *RRT* à été omis car ils n'ont aucun intérêt pour cette comparaison.

TAB. 2.7 – Résultats pour un environnement avec des passages de 20x20 et un espace des configurations de dimension 6.

2 Murs 4p0	Algorithme	Itérations	Nœuds	Temps
	LTRRT	3881	1973	329 s
	VISPRM	3820	7	232 s
	VISLT	1294	384	81 s
4 Murs 4p0	LTRRT	18273	7716	1755 s
	VISPRM	11954	19	1229 s
	VISLT	8288	1403	552 s
6 Murs 4p0	LTRRT	31972	11920	3329 s
	VISPRM	36359	116	5554 s
	VISLT	11771	1756	894 s
8 Murs 4p0	LTRRT	51016	15218	5357 s
	VISPRM	40606	162	9514s
	VISLT	22084	2634	2006 s

2.3.4.4 Analyse

Diminution de l'ajout du nombre de nœud

Pour montrer l'intérêt de notre méthode, considérons un problème avec une configuration de départ et une configuration d'arrivée situées dans deux pièces totalement isolées l'une de l'autre. Dans ce cas il n'existe aucune solution car les configurations de départ et d'arrivée sont dans des composantes disjointes (le problème est représenté sur la figure 2.18). Comme il

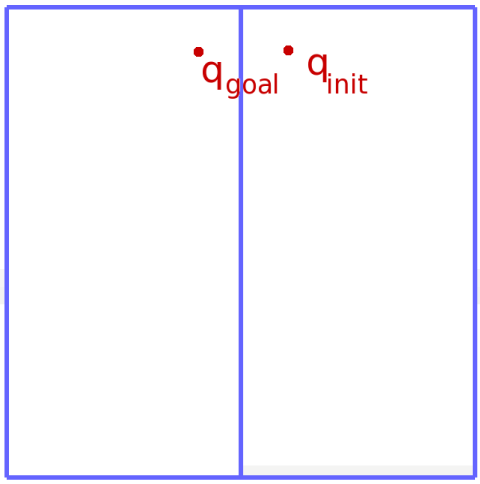


FIG. 2.18 – Un problème sans solution.

n'existe aucune solution, le temps de calcul des algorithmes a été limité à 10000 secondes afin d'arrêter la recherche probabiliste. La figure 2.19 montre le nombre de nœuds créés par chaque algorithme par rapport au nombre d'itérations effectuées durant le temps imparti. Le problème a été testé par quatre algorithmes différents : RRT, $LTRRT$, $VISPRM$ et $VISLT$. Sur la figure 2.19, on peut voir que le nombre de nœuds ajoutés au graphe augmente de manière constante lorsque les méthodes RRT (en bleu) et $LTRRT$ (en vert) sont utilisées. Cela est dû au fait qu'il n'y pas de contrôle sur l'ajout d'un nœud au graphe, en dehors des tests de collision et du choix de la densité de la roadmap.

À la différence des deux premières méthodes, nous pouvons voir que le $VISLT$ ajoute de moins en moins de nœuds au fur et à mesure que le nombre d'itérations augmente. En effet, la contrainte de distance par rapport à la racine de la composante connexe permet de réduire la densité du graphe et d'accélérer l'exploration de l'espace des configurations. Lorsque les limites de l'environnement sont atteintes, le nombre de nœuds ajoutés diminue naturellement. Cela permet de définir un critère d'arrêt de l'algorithme basé sur le nombre de nœuds ajoutés (on peut arrêter les calculs lorsque l'on insère peu de nœuds). La réduction de la densité du graphe a comme inconvénient de réduire les probabilités de trouver un passage étroit, mais cet inconvénient est pallié par l'ajout des arbres locaux de part et d'autre de ces passages. Une autre manière de pallier cet inconvénient pourrait être de modifier dynamiquement le coefficient c_{dist} appliqué aux comparaisons de distances lors de l'ajout de nœuds éclaireurs, lorsque le nombre de nœuds ajoutés diminue. Ces courbes montrent donc bien que l'ajout de nœuds au graphe de la méthode des arbres locaux de visibilité est fait uniquement lorsque c'est utile avec pour conséquence la réduction du nombre d'itérations nécessaires pour trouver une solution. Sur la figure 2.19 on peut voir que la méthode $VISPRM$ n'a créé aucun nœud durant le temps de calcul imparti. C'est le comportement normal de cette méthode, les nœuds initial et final fournissant

le maximum de visibilité pour ce problème puisqu'ils couvrent tout l'espace libre. Aucune connexion n'est possible entre les deux composantes, et le graphe ne peut pas être étendu.

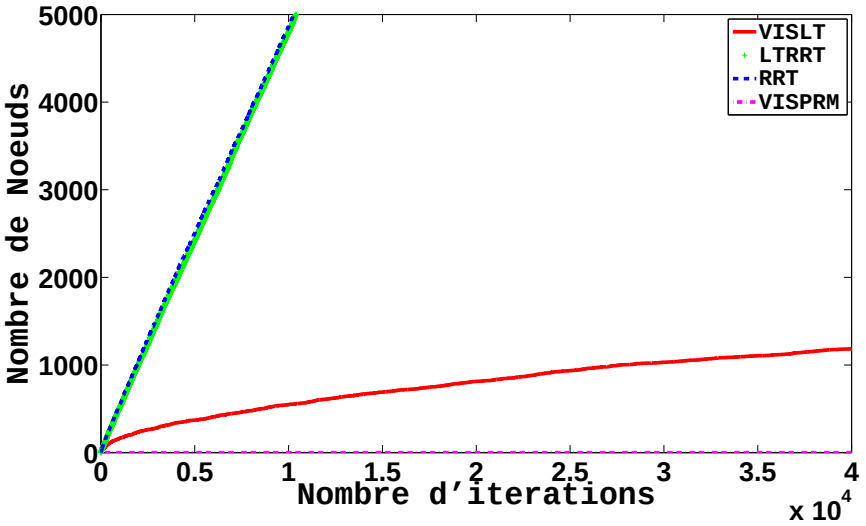


FIG. 2.19 – Augmentation du nombre de nœuds par rapport au nombre d'itérations sur le problème de l'impasse représenté figure 2.18

Les performances selon la difficulté des passages étroits

La courbe 2.20 montre l'effet de la difficulté croissante (due au rétrécissement de la taille des passages étroits) sur le nombre d'itérations nécessaires à la résolution, dans le cas d'un environnement avec seulement deux passages (et donc trois pièces - figure 2.14) et un espace des configurations de dimension 3 (X, Y, θ). On peut voir que pour chaque taille de passage, notre méthode parvient à résoudre le problème en moins d'itérations que les trois autres méthodes. On peut aussi remarquer que plus la taille diminue (en partant de la gauche et en allant vers la droite), plus la différence en nombre d'itérations entre les méthodes s'accroît. En regardant la pente des courbes, on voit que même si la méthode des arbres locaux de visibilité donne les meilleurs résultats, la courbe croît de la même manière que les deux autres avec la complexité du problème. Ce n'est pas le cas lorsque qu'on augmente le nombre de passages étroits. Pour le problème représenté figure 2.15, la courbe 2.21 représente les résultats en terme de nombre d'itérations en fonction de la taille des passages, comme pour le cas précédent. Sur cette courbe, le nombre d'itérations de l'algorithme *VISLT* croît beaucoup moins vite que pour les deux autres. Il trouve et explore plus rapidement les passages étroits et semble donc plus adapté dans des problèmes avec une succession de pièces séparées par de petites ouvertures.

La dernière courbe permet donc de mieux mettre en évidence les différences et les

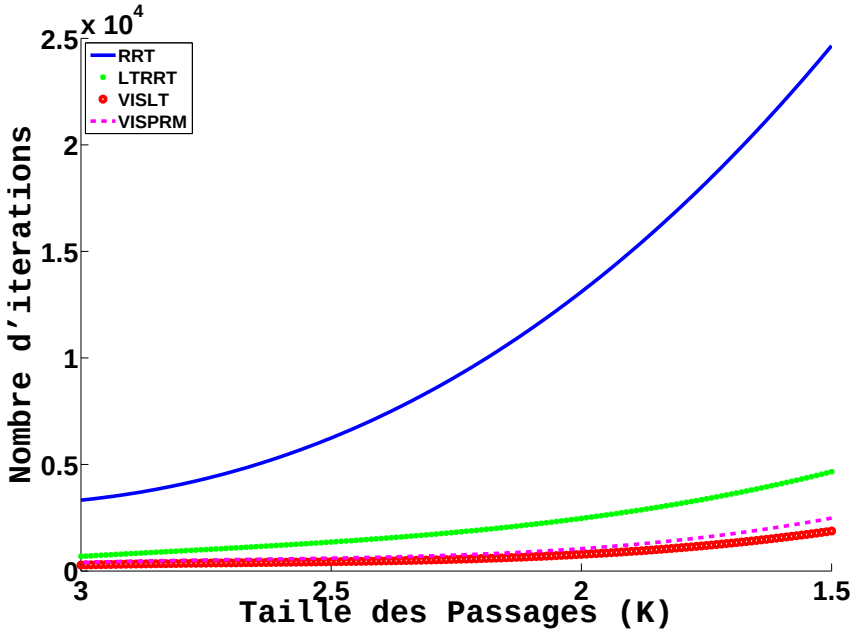


FIG. 2.20 – Influence de la taille du passage étroit dans le cas d'un environnement avec deux passages et un espace des configurations de dimension 3.

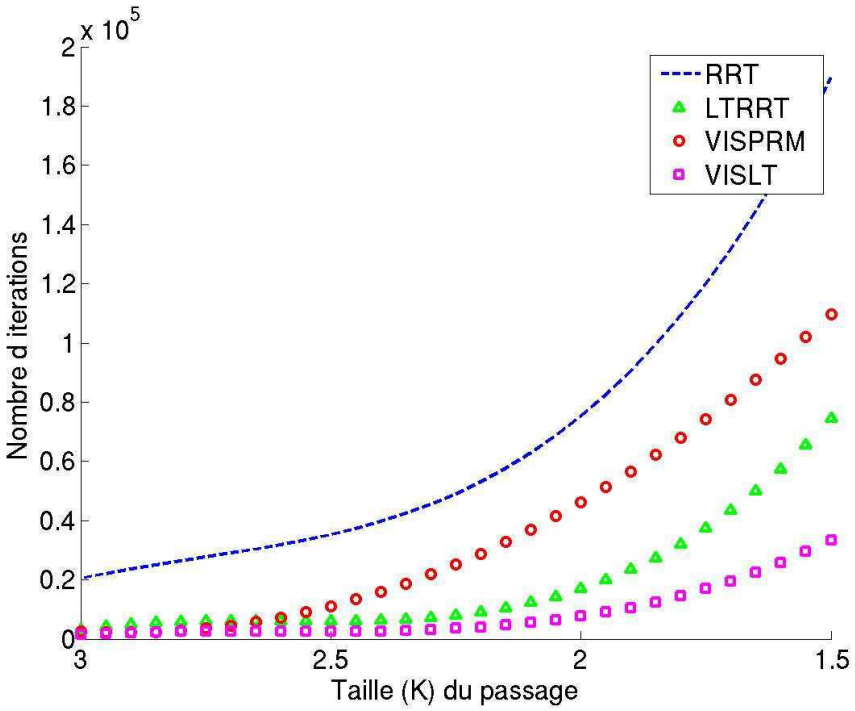


FIG. 2.21 – Influence de la taille du passage étroit dans le cas d'un environnement avec quatre passages et un espace des configurations de dimension 3.

similitudes entre les différents algorithmes comparés. On voit que les passages étroits sont le point faible des graphes de visibilité car ils réduisent drastiquement le domaine de visibilité de chaque nœud gardien et donc le domaine de visibilité commun entre chaque composante connexe du graphe. Ce fait est représenté sur la courbe par la pente élevée de la méthode des graphes de visibilité. Ce problème apparaît aussi avec la méthode *RRT* basique. Pour cette méthode, nous n'avons pas permis la diffusion à partir de la configuration finale. Bien qu'il ait moins de difficultés en premier lieu (pour les problèmes avec une passage plus grand), le nombre d'itérations nécessaires augmente lorsque la taille des passages diminue jusqu'à être supérieur à celui du *VISPRM*. Bien que la recherche des passages étroits soit moins difficile pour l'algorithme *RRT* que pour le *VISPRM*, la combinaison de ce problème avec une exploration beaucoup plus dense de l'espace des configurations libres induit une difficulté plus grande pour l'algorithme *RRT*. L'algorithme *LTRRT* souffre beaucoup moins de la recherche des passages étroits, ce qui lui permet d'être plus performant que les deux précédents algorithmes. Mais il effectue toujours une exploration dense. Enfin, l'algorithme *VISLT* profite des avantages d'une recherche rapide des passages étroits et d'une exploration moins dense, ce qui lui permet d'être plus performant que les autres algorithmes. Cette analyse peut aussi s'appliquer à la courbe 2.20, dans une moindre mesure.

En comparant les courbes 2.20 et 2.21, on remarque que les différences entre les algorithmes sont plus marquées dans la seconde. Cela nous permet de dire que le nombre de passages étroits a une grande influence sur la résolution et les algorithmes comparés. La différence est plus visible avec le *VISPRM*, ce qui confirme que les passages étroits sont son point faible. Seul le *VISLT* semble peu affecté par le nombre de passages étroits.

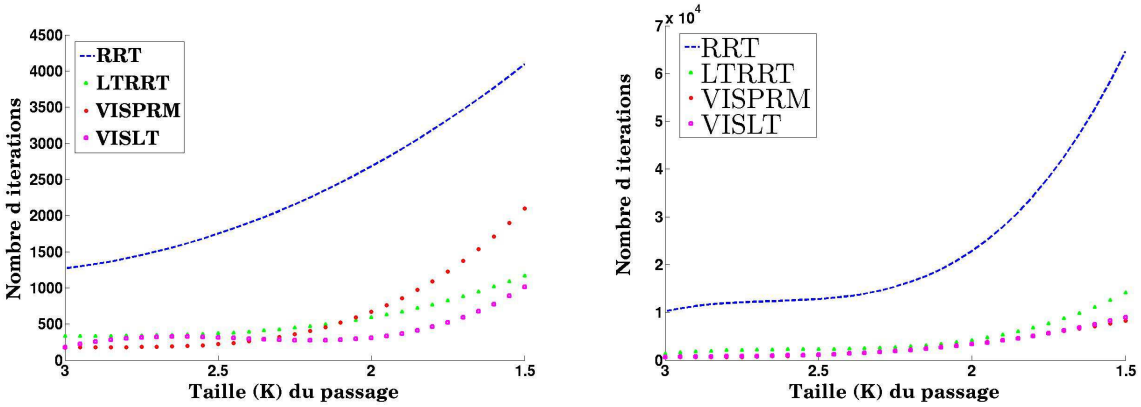


FIG. 2.22 – Influence de la taille du passage étroit dans le cas d'un environnement avec deux passages (à gauche) et dans la cas d'un environnement avec quatre passages (à droite) et un espace des configurations de dimension 2.

Enfin si on compare les différences entre les cas 2D (courbes 2.22) et 3D (courbes 2.20 et 2.21) avec les différences entre les cas avec deux passages (courbes 2.20 et 2.22 (gauche))

et quatre passages (courbes 2.21 et 2.22 (droite)), il est clair que les premières sont moins importantes que les secondes. Cette observation confirme le fait que le nombre de passages étroits a plus d'influence sur la résolution que la dimension de l'espace des configurations.

La taille des passages étroits joue donc un rôle important lors de la résolution de problème du type "succession de pièces séparées par des passages étroits". Bien que les algorithmes sur lesquels l'algorithme *VISLT* est basé soient très dépendants de la taille de ces passages, leur combinaison permet d'avoir un algorithme *VISLT* très peu affecté. Mais cette caractéristique de l'environnement n'est pas la seule à influer sur le temps de résolution des problèmes : le nombre de passages étroits a aussi un effet important.

Les performances selon le nombre de passages étroits

Après une analyse des performances de l'algorithme *VISLT* en fonction de la taille des passages étroits, il est intéressant de regarder l'influence du nombre de passages. Ce nombre reflète la fragmentation de l'environnement, c'est-à-dire le nombre de composantes faiblement connexes de celui-ci. Sur la figure 2.23, on peut comparer les performances des trois algorithmes (*LTRRT*, *VISPRM* et *VISLT*). Le contrôle de la croissance des arbres locaux fait que l'algorithme *VISLT* demande moins de temps et d'itérations pour trouver une solution par rapport aux deux autres, ainsi que de nombre de nœuds ajoutés dans le graphe par rapport à l'algorithme *LTRRT*. La pente des différentes courbes montre que pour chaque ajout d'un passage étroit à traverser, le coût est moindre pour le *VISLT* que pour les deux autres (en temps et en nombre d'itérations). Là encore, le *VISLT* profite de la caractéristique principale du *VISPRM*, à savoir un nombre de nœuds réduit, par rapport au *LTRRT*, ainsi que de celle du *LTRRT* - une meilleure probabilité de trouver les passages étroits - par rapport au *VISPRM*, ce qui fait du *VISLT* une bonne alternative par rapport aux deux autres algorithmes pour le type d'environnement considéré.

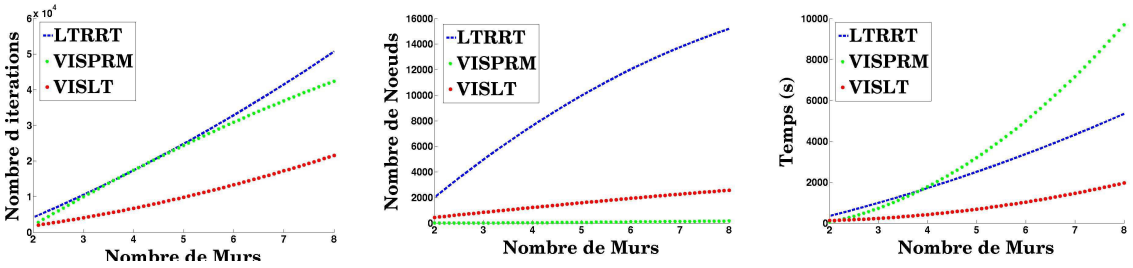


FIG. 2.23 – Les performances en fonction du nombre de passages étroits à traverser.

Évolution du nombre de composantes connexes du graphe

Les trois algorithmes comparés dans cette analyse (*LTRRT*, *VISPRM* et *VISLT*) reposent tous sur la création, le contrôle et l'utilisation de multiples composantes connexes pour

construire leur graphe et ainsi résoudre le problème de planification qui leur est posé.

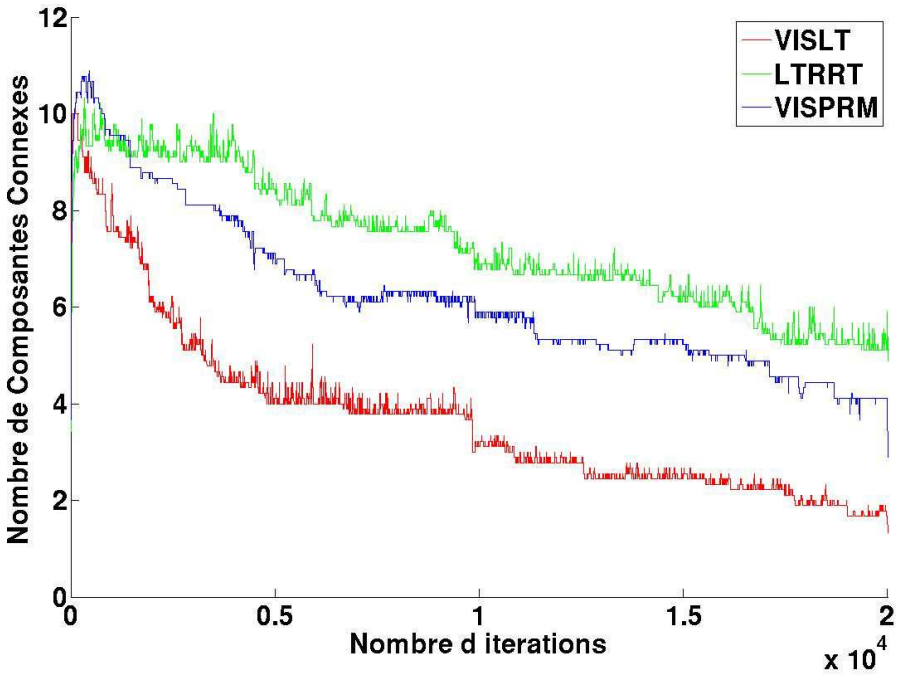


FIG. 2.24 – Évolution du nombre de composantes connexes.

Sur la courbe 2.24, on peut voir l'évolution du nombre de composantes connexes pour chaque algorithme au cours d'une requête de planification de mouvement. Les tests ont été effectués sur un environnement de type "murs" (comme sur la figure 2.17), possédant une succession de 8 murs. Chaque requête a été stoppée après 20000 itérations et les courbes sont une moyenne sur dix requêtes pour chaque algorithme. Cette courbe montre que la méthode *VISLT* crée moins de composantes connexes et est capable de les connecter beaucoup plus vite que le *VISPRM* et le *LTRRT*. On observe mieux cette vitesse de connexion lors des premières itérations : la courbe rouge (*VISLT*) décroît plus rapidement que les courbes verte (*LTRRT*) et bleue (*VISPRM*).

Le bruit que l'on peut observer sur chacune des courbes est dû au fait que des composantes connexes sont créées sans cesse, puis immédiatement connectées à une autre composante après très peu d'itérations. Ces composantes, que nous appellerons composantes éphémères, sont situées dans de petites zones qui ne sont pas réellement difficiles d'accès, mais qui ne peuvent pas être directement liées à une autre composante connexe, à cause notamment de la proximité d'obstacles (près d'un mur par exemple). Dans notre exemple, cela est dû en grande partie aux rotations et à la méthode locale linéaire utilisée pour lier deux configurations. Le fait que la connexion entre un échantillon aléatoire nouvellement créé et une composante connexe existante ne soit testée qu'avec le nœud le plus proche dans cette composante accentue

encore l'effet « composante éphémère ». En effet, si le nœud le plus proche n'est pas forcément accessible pour l'échantillon, d'autre nœuds plus éloignés peuvent l'être. La composante qui est créée à partir de cette échantillon pourrait donc être connectée en quelques itérations.

La courbe du *VISPRM* semble moins bruitée que les autres courbes car l'algorithme a plus de mal à connecter les différentes composantes connexes nouvellement créés avec les autres. Au contraire, les algorithmes *VISLT* et *LTRRT* peuvent créer et connecter de nouvelles composantes relativement facilement, ce qui est représenté par le bruit plus dense de leurs courbes respectives. Mais dans le cas du *LTRRT*, le nombre de nœuds ajoutés au graphe avant qu'il y ait une connexion entre deux composantes connexes est plus important, ce qui explique que le nombre de composantes du *VISLT* est toujours inférieur à celui du *LTRRT*.

On peut aussi remarquer le fait que la courbe du *VISLT* est plus bruitée au début qu'à la fin. Cela montre bien les capacités d'exploration de l'algorithme : la plupart des petites zones plus ou moins accessibles sont trouvées et connectées rapidement au reste du graphe, de même que les zones plus larges ou plus isolées, ce qui réduit le nombre de composantes connexes du graphe. Le *VISPRM* crée plus de composantes dans les zones libres larges et faciles à trouver, mais ajoute moins de nœuds situés dans les petites zones, ce qui explique que sa courbe soit moins bruitée à la fin. Ceci est principalement dû au fait que ces petites zones sont plus facilement connectables dans le cas du *VISPRM* et donc présentent peu d'intérêt en terme de domaine de visibilité. Quand au *LTRRT*, sa courbe est aussi bruitée au début qu'à la fin, ce qui est dû au grand nombre de nœuds ajoutés au graphe et à sa densité.

Résolution des problèmes posés par les arbres locaux

Dans les parties 2.2.1 et 2.3, nous avons parlé de plusieurs points cruciaux à résoudre et dont dépendent les performances des algorithmes à base d'arbres locaux. Ces points concernaient la création de nouvelles composantes connexes, leur croissance et l'arrêt de leur croissance. Nous avons vu dans les exemples précédents que chacun d'entre eux à été résolu dans notre implémentation des arbres locaux de visibilité. Le premier point (la création d'une nouvelle composante connexe) est résolue par la partie "*VISPRM*" de l'algorithme : une nouvelle composante connexe n'est créée que lorsqu'un nouvel échantillon ne peut-être relié à aucune autre composante connexe du graphe. C'est-à-dire quand un nœud gardien est créé. La composante s'étend jusqu'à ce que les limites de l'environnement local (zone directement visible par la composante connexe) soient atteintes. Le nombre de nœuds ajoutés à cette composante diminue ensuite progressivement, favorisant ainsi une exploration large et rapide de l'espace des configurations plutôt qu'une exploration dense (contrairement au *RRT* et au *LTRRT*). Ce contrôle de la croissance répond à la seconde problématique posée par les arbres locaux. La croissance est quasi-nulle lorsque l'environnement local est totalement occupé par la composante connexe. En deçà d'un certain seuil, l'ajout de nœuds pour cette composante peut-être explicitement stoppé. Une autre manière d'arrêter la croissance d'une composante est

de la fusionner avec une autre, lorsqu'un nœud connecteur le permet. Les deux composantes fusionnées grandissent ensuite en tant que nouvelle composante. Ces deux façons de stopper la croissance d'une composante répondent à la dernière problématique posée par les arbres locaux.

2.4 Conclusion et Perspectives

Tout au long de ce chapitre nous avons décrit une méthode permettant de combiner avantageusement les principes des arbres locaux et des graphes de visibilité. Cette méthode est basée sur le principe du *RRT* et utilise les arbres locaux afin d'accélérer l'exploration de l'espace des configurations et de trouver les passages étroits plus facilement. L'intégration des contraintes de visibilité permet de limiter le nombre de nœuds créés dans le graphe et de maximiser la couverture de l'espace des configurations. Cette méthode est conçue pour mieux fonctionner dans des environnements formés d'une succession de pièces vides et larges séparées par de courts passages étroits. L'algorithme des arbres locaux de visibilité est un bon compromis entre un graphe dense qui serait coûteux à construire et un graphe trop petit pour représenter correctement la connectivité d'un environnement possédant des passages étroits.

On a vu dans ce chapitre qu'un des paramètres importants du $VISLT$ concerne l'ajout de nœuds éclaireurs au travers du coefficient c_{dist}. Il serait intéressant d'étudier l'effet de ce coefficient sur la vitesse de résolution, ainsi que de pouvoir le modifier de manière dynamique au cours de la résolution et selon les composantes connexes. Les environnements étudiés dans ce chapitre ont la particularité d'avoir un espace de travail qui a la même topologie que l'espace des configurations (succession de pièces séparées par des passages étroits). Il serait intéressant d'étudier des exemples plus complexes avec des robots articulés mais qui présentent le même type de topologie dans l'espace des configurations. Cette amélioration des algorithmes de base permet d'adapter les méthodes de planification de mouvement à la locomotion humaine en environnement structuré. Elle a aussi pour but d'accélérer la planification automatique afin de pouvoir l'intégrer dans un système de planification interactive faisant intervenir un opérateur humain dans la recherche de chemin. C'est cette partie que nous allons aborder dans le prochain chapitre.

3

Le planificateur de mouvement interactif

3.1 Introduction

Dans beaucoup d'applications robotiques, telles que l'assemblage ou le désassemblage de pièces mécaniques, il est question de déplacer un objet dans des environnements simulés pour vérifier la faisabilité d'une tâche. Dans ces environnements simulés, il est possible de choisir entre un déplacement effectué manuellement par un opérateur humain via un périphérique (appelé périphérique d'interaction) lui permettant d'interagir avec l'environnement virtuel (tel qu'une souris, un clavier ou encore un bras à retour d'effort), ou un déplacement entièrement automatique à l'aide d'un algorithme de planification tel que ceux présentés au chapitre 2.

Dans ce chapitre, nous allons présenter un système basé sur l'interaction entre un utilisateur et un algorithme de planification de mouvement. L'utilisateur a pour but de guider la recherche d'un chemin pour un objet ou un robot entre une configuration initiale et une configuration finale. Il a pour cela à sa disposition un périphérique d'interaction qui peut être une souris 6D ou un périphérique haptique. Le domaine de la simulation haptique est une amélioration récente des périphériques d'interaction qui permet à un utilisateur d'avoir un mode de perception additionnel (par rapport à la combinaison classique des modes visuels et auditifs) sur un environnement virtuel [Burdea 2000; Fuchs 2006; Boulic et al. 2006]. Un périphérique dit haptique permet à l'utilisateur de ressentir les forces en jeu dans l'environnement virtuel (des forces de contact par exemple) et ainsi, dans un cadre de planification de mouvement, de tester des chemins libres de collision. L'une des applications - qui va être traitée dans ce chapitre

- est l'assemblage de pièces mécaniques lors d'un processus industriel de conception ou de maintenance [Galeano and Payandeh 2005].

Une autre alternative - qui s'avère moins chère et donne d'excellents résultats dans des processus de conception [Chen and Brown 2005; Su et al. 2008] - est l'utilisation d'une souris 6D en lieu et place d'un périphérique haptique. C'est pourquoi elle a été utilisée dans la plupart de nos tests (voir parties 3.2.3 et 3.2.4). Cependant, comme un périphérique haptique peut s'avérer nécessaire dans des environnements très complexes où la visualisation peut-être difficile et confuse (occultations, affichage 2D ...), nous en avons aussi utilisé un pour certains exemples.

Dans le domaine de la robotique, beaucoup de travaux sur la planification de mouvement pour des systèmes mécaniques dans des modèles numériques existent. Des exemples d'applications des techniques de planification de mouvement dans des cas réels d'utilisation ont déjà donné de bons résultats dans le cas d'opérations de maintenance dans les centrales nucléaires [Siméon et al. 2001; Siméon et al. 2002] ou encore des applications de gestion du cycle de vie d'un produit (*Product Lifecycle Management* [Ferré et al. 2005; Laumond 2006]).

Parmi les deux types de méthodes les plus fréquemment utilisées en planification de mouvement (*PRM* et *RRT*, voir 2.1.1), nous avons choisi de nous baser sur une méthode de type RRT, car elle est plus adaptée au type de problèmes traités, à savoir des problèmes à requête unique, où la construction du graphe doit se faire en temps limité pendant la recherche de chemin. Dans le cas d'environnements très contraints [Ferré and Laumond 2004] il existe des méthodes très efficaces, mais ce n'est pas le cas pour des environnements combinant des zones sans obstacles larges et des passages étroits dans un espace des configurations avec un grand nombre de dimensions. Un exemple typique est un objet passant à travers un tunnel très étroit. Les algorithmes classiques de planification de mouvement ont beaucoup de mal à trouver l'entrée du passage étroit (voir par exemple [Boor et al. 1999; Wilmarth et al. 1999b; Amato et al. 1998; Sun et al. 2005; Strandberg 2004]), car il leur faut explorer toutes les dimensions de l'espace des configurations. Mais lorsque l'arbre a commencé à se développer à l'intérieur du passage, il progressera rapidement jusqu'à la solution. Il existe par ailleurs des méthodes efficaces récentes pour accélérer cette progression [Dalibard and Laumond 2009].

Les algorithmes aléatoires peuvent avancer rapidement à l'intérieur des passages étroits mais n'ont pas de vision globale de la solution. Au contraire, un utilisateur humain trouvera facilement un chemin dans un environnement peu contraint ou l'entrée d'un passage étroit. Il a une vision plus globale du problème, mais éprouve des difficultés à exécuter des chemins demandant une grande précision. En conclusion, un utilisateur humain et un algorithme de planification de mouvement peuvent travailler de manière complémentaire sur ce type de problèmes. La complémentarité entre humain et algorithme est l'idée essentielle de ce travail. La contribution principale de cette partie est de permettre la coopération entre les deux afin d'améliorer la recherche de solution et le guidage d'un opérateur humain à l'aide d'algorithmes

de planification de mouvement. Ce travail s'insère dans le cadre d'un projet que nous allons présenter. Nous rappellerons les travaux déjà effectués sur la même thématique avant de décrire la méthode développée et son implémentation. Nous finirons en présentant les avantages de la méthode à travers différents exemples d'utilisation.

3.1.1 Le projet AMSI

Les travaux décrits dans ce chapitre ont été réalisés dans le cadre du projet « Algorithmique du Mouvement et Simulation Interactive » (AMSI) de 2007 à 2009. Le projet AMSI est un projet financé par l'Agence Nationale de la Recherche (ANR) dans le cadre du programme « Technologies Logicielles ». Le projet portait sur l'assemblage mécanique au sein de maquettes numériques. Il s'agit de concevoir et de réaliser des solutions de recherche de chemins dans ces modèles permettant de prendre en compte les actions d'un opérateur humain en temps réel. Contrairement aux méthodes entièrement automatiques, cela permet d'imposer des contraintes « métier » sur les chemins produits, contraintes qui font partie du savoir-faire de l'utilisateur (préférer un passage moins dangereux, passer plus loin d'une certaine pièce ...). L'idée de base du projet est de combiner deux approches : l'une entièrement manuelle et réalisée grâce à des bras à retour d'effort (ou bras haptiques) et des systèmes de réalité virtuelle et l'autre, entièrement automatique, est réalisée grâce à des algorithmes de planification de mouvement probabilistes.

Le projet regroupait quatre partenaires : deux laboratoires (le CEA-LIST et le LAAS-CNRS) et deux start-ups (Haption et Kineo CAM). Pour atteindre l'objectif, trois approches ont été envisagées :

- Un première approche séquentielle, dans laquelle l'opérateur et la partie algorithmique agissent séparément dans deux étapes distinctes et successives. L'opérateur agira alors sur le résultat fourni par l'algorithme afin de l'améliorer.
- Une seconde approche semi-interactive où l'opérateur peut influer ponctuellement sur la recherche automatique. Les parties manuelle et automatique agiraient donc alternativement, mais en étant toujours distinctes l'une de l'autre.
- Une troisième approche, dite interactive, où l'opérateur et l'algorithme coopèrent dans une même recherche, en temps réel, en même temps et sans distinctions entre les deux

La méthode décrite dans ce chapitre se place dans le cadre de l'approche interactive.

3.1.2 Travaux précédents

L'idée de prendre en compte les actions d'un utilisateur lors d'une recherche de chemin a déjà été étudiée par différents auteurs spécialisés dans le domaine de la planification de mouvement. Dans [Bayazit et al. 2000], les auteurs font travailler un opérateur en commun

avec un algorithme de planification de mouvement automatique. Ils montrent que les techniques de planification probabilistes peuvent être utilisées pour transformer un chemin utilisateur approximatif en collision avec l'environnement en un chemin sans collisions. L'idée est de déformer les parties du chemin qui sont en collision en les poussant hors de l'espace obstrué.

Dans [Vazquez and Rosell 2007; Rosell et al. 2008] est introduite l'utilisation d'une technique de planification de mouvement basée sur des fonctions harmoniques permettant de générer des forces de guidage qui aident l'utilisateur à se déplacer dans un environnement virtuel. L'idée de base est de calculer un tunnel solution par décomposition cellulaire de l'environnement qui connecte les configurations initiale et finale. Dans un second temps, deux fonctions harmoniques sont calculées sur l'espace des configurations pour trouver un chemin-guide.

Un algorithme de type RRT incluant des heuristiques basées sur l'étude de l'espace de travail est présenté dans [Ladezeve et al. 2008; Ladeveze et al. 2009]. Le but est de discrétiser l'espace de travail en utilisant un arbre de décomposition type octree non-équilibré. Ensuite les auteurs calculent un volume continu dans l'espace de travail libre entre les configurations initiale et finale en utilisant un $A^\star$. Dans une seconde étape, un chemin libre de collisions est calculée à partir du volume créé précédemment en utilisant une approche RRT.

D'autres travaux en dehors du champs d'application de l'assemblage mécanique existent. Par exemple, dans le domaine de la biologie, des applications étudient les performances d'un planificateur de mouvement utilisant un périphérique à retour d'effort pour prendre en compte des entrées d'un utilisateur [Bayazit et al. 2001].

Dans [He and Chen 2009] les auteurs étudient l'influence d'une intervention humaine sur la vitesse de convergence d'un planificateur de mouvement de type PRM lors d'une tâche de planification. L'utilisateur indique des configurations de passage obligatoires pour le robot en utilisant un périphérique à retour d'effort.

Dans tous ces travaux, on peut remarquer que l'interaction entre l'utilisateur et la recherche de chemin automatique est simplifiée en découplant l'algorithme en deux étapes distinctes : une étape où le planificateur de mouvement effectue la recherche seul et une étape ou l'utilisateur intervient seul. Dans ce chapitre, il sera question d'une méthode permettant de faire coopérer les deux parties dans une seule et même boucle d'interaction.

3.2 Le planificateur interactif

3.2.1 Périphériques interactifs

Depuis une décennie, de nouveaux types de périphériques d'interaction permettent aux utilisateurs d'ordinateurs d'interagir de manière plus naturelle avec les machines et les simulations virtuelles [Duriez et al. 2006; Barbe et al. 2007]. En effet, ces périphériques

ajoutent à leurs prédécesseurs plusieurs degrés de contrôle qui manquaient par rapport à la réalité : un contrôle en six dimensions - trois translations et trois rotations (ce que ne peut pas faire une souris normale) - et pour certains la sensation de retour d'effort, aussi appelée sens haptique. Ces périphériques permettent une manipulation plus fine et plus précise des objets, virtuels ou réels et un apport d'information supplémentaire à l'utilisateur sur ce qu'il se passe au delà du périphérique d'interaction. Une information qu'un simple affichage sur écran ne permet pas. L'utilisation de périphériques haptiques peut améliorer l'utilisation de la réalité virtuelle [Ye et al. 1999; Chabal et al. 2005]. De nombreuses applications utilisent aujourd'hui ces périphériques dans des domaines variés tels que l'apprentissage [Bayart et al. 2005; Bayart and Kheddar 2007], le désassemblage [Lecuyer et al. 2001], ou la chirurgie [Baumann and Clavel 1997].

Ces différents périphériques, tels que la "souris 6D" et le bras haptique (figure 3.1), peuvent être utilisés pour améliorer la planification de mouvement avec une interaction utilisateur plus naturelle, permettant ainsi d'accélérer la recherche de chemin. Dans le cas d'une souris 6D, l'utilisateur peut agir sur le déplacement en translation et en orientation des objets qu'il contrôle. Un des inconvénients dans ce cas est que l'utilisateur ne reçoit que des informations visuelles au travers d'un écran et n'a aucune information haptique. Ces informations haptiques peuvent être nécessaires quand l'affichage ne permet pas de déterminer un chemin à suivre, typiquement lorsque l'objet déplacé doit entrer à l'intérieur d'un autre, ou lorsque les objets et les obstacles sont très complexes et que le chemin solution comporte de multiples rotations. Ces informations peuvent être fournies par un périphérique haptique tel que le Virtuose6D Desktop qui est un bras à six *ddl* qui peut transmettre un retour d'effort sur tous ses *ddl* et qui peut calculer et transmettre à l'utilisateur un effort sous forme de force ou de couple. Par exemple, l'homme utilise sa perception des efforts lorsqu'il exécute une opération d'insertion. Cette information est importante mais difficile à retranscrire fidèlement au travers d'un système informatique.

Dans la suite de ce chapitre, le terme périphérique interactif sera utilisé pour désigner tous types de périphérique (souris ou bras haptique). Si le périphérique utilisé est une souris sans retour d'effort, alors des forces de retour virtuelles sont calculées à partir du mouvement de la souris. Sinon, les forces de retour sont directement lues à partir du périphérique interactif. On appellera pseudo-forces les retours des deux types de périphériques interactifs. Ces pseudo-forces prennent aussi en compte les couples retournés par les périphériques interactifs.

3.2.2 Principe

3.2.2.1 Fonctionnement général du planificateur interactif

Considérons un humain (l'opérateur) utilisant un périphérique interactif pour contrôler le déplacement d'un robot dans un environnement virtuel. La méthode que nous proposons a pour

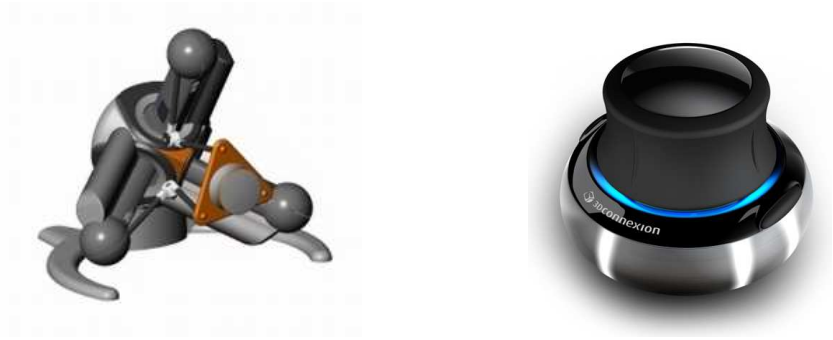


FIG. 3.1 – Le Virtuose6d Desktop de Haption (à gauche) et le SpaceNavigator de 3DConnexion Company (à droite).

but d'améliorer le guidage de l'opérateur à l'aide d'algorithmes de planification de mouvement automatique.

L'opérateur déplace un objet, que nous appellerons avatar, avec le périphérique interactif, objet qui peut-être :

- un objet "volant" qui peut se déplacer librement selon ses 6 *ddl*. C'est le cas dans les problèmes d'assemblage/désassemblage,
- un robot humanoïde contrôlé par le pelvis ou les épaules,
- un bras manipulateur dont on contrôle uniquement l'organe terminal. L'opérateur désire seulement guider l'organe terminal du robot sans en contrôler tous les *ddl*.

Notre méthode prend en compte deux principales contraintes :

- la direction de mouvement de l'opérateur qui doit diriger le développement du graphe.
- le planificateur qui doit retourner à l'opérateur des informations utiles sur l'environnement (informations qui peuvent être haptiques ou visuelles) concernant la proximité des obstacles et si la zone a déjà été explorée.

La méthode de planification interactive présentée est basée sur le *RRT* et se compose de deux boucles principales qui fonctionnent en parallèle (figure 3.2). Dans la première boucle (à droite sur la figure, dans la boîte « Algorithme Interactif »), le planificateur de mouvement explore l'espace des configurations libres et recherche une solution de façon automatique. L'échantillonnage et l'étape d'extension de l'algorithme *RRT* de base sont modifiés afin de prendre en compte l'interaction de l'utilisateur via le périphérique interactif. Dans la seconde boucle (à gauche, « Périphérique Interactif »), l'opérateur déplace le périphérique interactif comme il le souhaite pour déplacer l'avatar dans la scène virtuelle. L'interaction de l'opérateur, qui va se traduire par une pseudo-force notée F_u, permet de déplacer un avatar dans l'environnement virtuel avec la perception du retour d'effort dans le cas d'un bras haptique et ces informations sont transmises à l'algorithme interactif (*IRRT*). Celui-ci calcule une pseudo-force F_a à partir de la position de l'avatar. F_a permet d'influencer le développement du graphe et sert en même temps à calculer la pseudo-force F_s retournée par l'algorithme. La pseudo-force

F_s est perçue comme une perturbation par l'utilisateur, mais lui sert en même temps de guide dans des zones étroites. Elle est appliquée au périphérique interactif ou directement à l'avatar selon le type de périphérique.

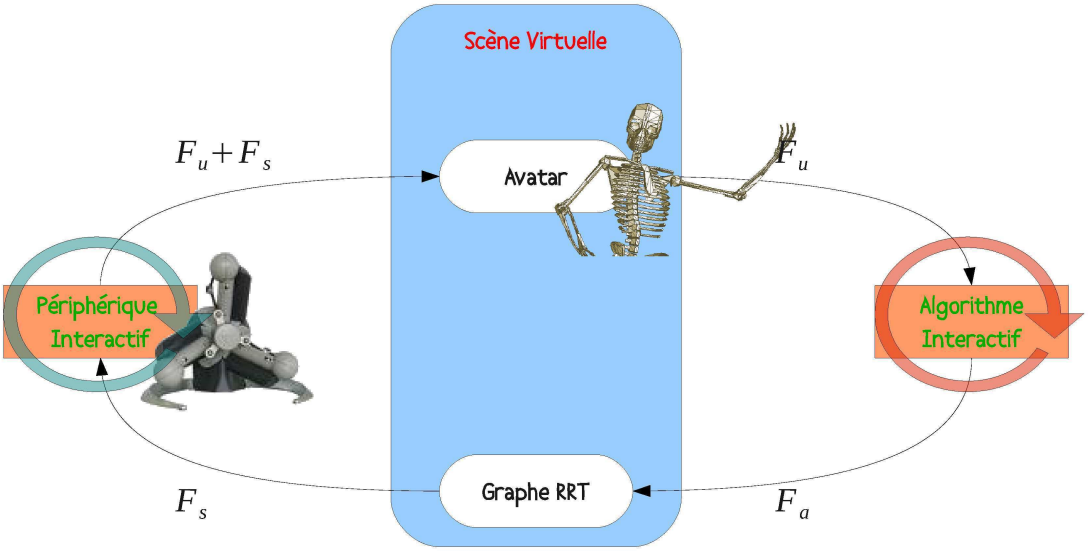


FIG. 3.2 – Boucle générale d'interaction entre l'algorithme et l'utilisateur via la scène virtuelle

Il est évident que l'utilisateur et l'algorithme vont avoir un fonctionnement qui leur est propre, aussi nous pouvons distinguer trois modes de fonctionnement pour cette méthode :

- L'utilisateur déplace son périphérique interactif plus rapidement que le graphe ne se développe car il sait où il souhaite se diriger, dans ce cas l'algorithme interactif poursuit la position de l'avatar dans la scène virtuelle. On suppose alors que l'intention de l'utilisateur est prépondérante. Si la solution est évidente pour l'utilisateur, alors l'algorithme suit ses déplacements.
- L'utilisateur ne bouge pas le périphérique : l'algorithme affiche des informations visuellement ou haptiquement sur des directions préférentielles à suivre. Il continue en même temps à explorer l'environnement de manière automatique, afin de pouvoir mieux informer l'opérateur. C'est la situation qui se produit lorsque l'utilisateur ne sait plus comment faire progresser l'avatar. On se ramène alors à une recherche automatique si ce mode de fonctionnement est prépondérant.
- L'utilisateur déplace le périphérique lentement : l'algorithme développe son graphe dans

la direction du mouvement de l'utilisateur tout en explorant d'autres zones proches en dehors de l'influence de l'utilisateur pour acquérir des informations supplémentaires. Il y a une utilisation des deux types de recherche de solution.

L'opérateur et l'algorithme ne travaillent pas forcément en même temps, aussi une synchronisation est faite entre les deux boucles de fonctionnement, qui travaillent à des fréquences différentes. Cette différence de fréquence n'est pas un problème car chacune des boucles est non-bloquante. Les seules informations partagées entre les deux boucles sont les pseudo-forces. L'algorithme continuera à lire F_u et à mettre à jour les pseudo-forces qui le concerne (F_a, F_s) quel que soit l'état de la boucle du périphérique interactif. De la même façon la boucle du périphérique continuera de fonctionner sans attendre les mises à jour des pseudo-forces.

3.2.2.2 Le système de planification de mouvement interactif

L'idée du *IRRT* est de profiter des capacités de l'opérateur et de l'ordinateur en même temps pour résoudre un problème de planification de mouvement dans un environnement virtuel. Le pseudo-code est donné dans l'algorithme 3.

Algorithm 3 Le RRT Interactif

```
T(q_init)
for i = 0 to N do
  q_rand ← SAMPLED_CONFIG(F_a, F_u)
  q_near ← NEAREST_NODE(q_rand, T)
  L_near ← NEAREST_NEIGHBORS(q_user, T)
  if CONNECT(T, q_rand, q_near, q_new) then
    Add_Vertex(T, q_new)
    Add_Edge(T, q_near, q_new)
    Update_Labels(T, q_near, q_new)
    COM ← Compute_CenterOfMass(L_near)
    F_a ← Compute_FAlgo(COM, q_user) ;
    F_u ← Compute_FUser(q_user) ;
  end if
end for
```

Algorithm 4 SAMPLED_CONFIG(F_a,F_u)

```
F_r ← GET_FORCE(F_a, F_u)
GRAM_SCHMIDT_PROCESS(F_r)
GAUSSIAN_SHOOT()
```

Les étapes principales de cet algorithme sont les suivantes :

- Calcul d'une pseudo-force attractive F_a à partir de l'arbre courant en utilisant les données contenues dans le graphe (ses nœuds) pour influencer le mouvement de l'utilisateur au travers du périphérique interactif et d'indices haptiques et/ou visuels.
- Calcul d'une pseudo-force d'interaction F_u à partir de la force appliquée par l'utilisateur sur le périphérique pour prendre en compte son intention.
- Échantillonnage d'une configuration à partir d'une pseudo-force $F_r = f(F_u, F_a)$: il faut choisir une manière efficace de combiner l'intention de l'utilisateur avec la recherche automatique de l'algorithme.
- Choix du plus proche voisin dans le graphe.
- Extension du graphe.
- Mise à jour des données des nœuds en fonction du résultat de l'itération courante.
- Envoi d'informations à l'opérateur.

Calcul de la pseudo-force attractive

Dans une première étape, comme pour un algorithme RRT classique, il nous faut échantillonner une nouvelle configuration afin de la tester. Il nous faut aussi définir l'influence de l'utilisateur sur le développement du graphe. C'est nécessaire pour pouvoir rendre le développement indépendant dans le cas d'un blocage de l'utilisateur lors de la recherche de chemin. Ainsi nous pourrons récolter des informations sur les zones environnant le passage de l'opérateur pour trouver des chemins alternatifs ou des directions préférentielles. Les informations récoltées seront ensuite transmises à l'utilisateur sous forme visuelle et haptique. Dans le cas d'une itération de la boucle générale de planification interactive, l'échantillonnage se fait à partir de plusieurs sources de données. Dans un premier temps, l'algorithme va lire les données fournies par le périphérique interactif afin de connaître la position de l'utilisateur dans la scène virtuelle. Une fois cette position connue, les p nœuds les plus proches de cette position sont choisis (figure 3.3-1, *NEAREST_NEIGHBORS* dans l'algorithme 3). Le centre de masse de ces nœuds est calculé en utilisant les données contenues dans les nœuds en tant que coefficients (le calcul de ces données - appelées étiquettes - est décrit dans la suite du chapitre) comme sur la figure 3.3 (2 et 3). La pseudo-force attractive, que nous noterons F_a, est donnée par le vecteur ayant pour origine la position de l'opérateur dans la scène virtuelle et passant par le centre de masse calculé auparavant. L'intensité de cette force est égale à la distance séparant la position de l'opérateur du centre de masse (figure 3.3-4).

Calcul de la pseudo-force résultante

Comme nous l'avons vu au début du chapitre, l'idée principale de la méthode de planification interactive présentée est que l'opérateur doit diriger le développement de l'arbre. Il faut donc définir l'influence de l'opérateur. Cela est fait en lisant les données fournies par le périphérique

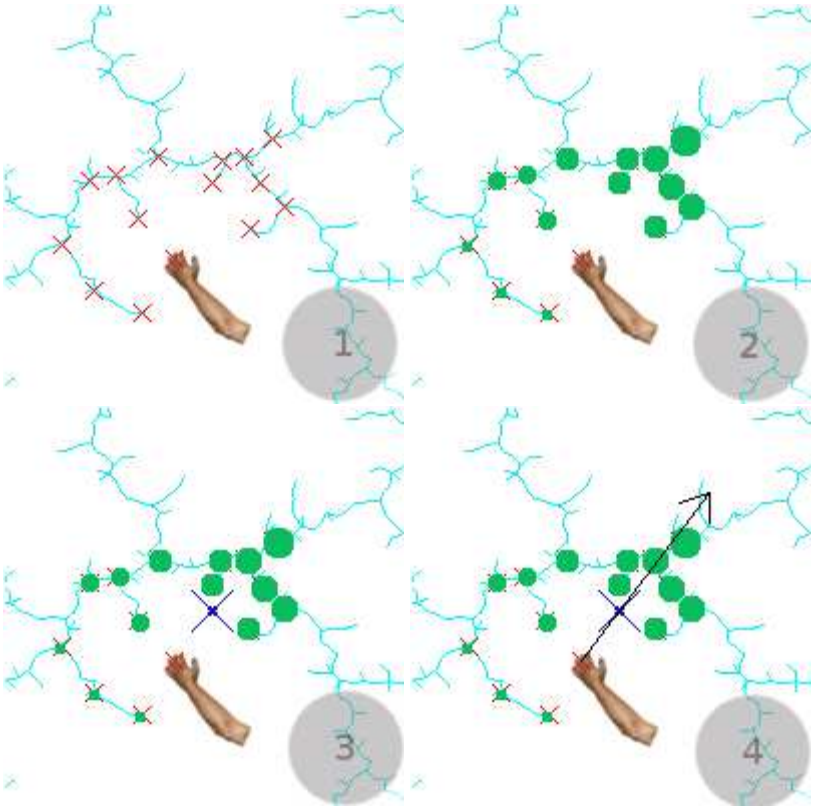


FIG. 3.3 – Calcul de la pseudo-force F_a.

interactif pour en déduire le mouvement de l'opérateur et donc son intention. Cette intention est traduite par la pseudo-force F_u récupérée dans l'algorithme 3 par $Compute\_FUser$. Ensuite, grâce à la pseudo-force attractive F_a calculée précédemment par l'algorithme, une pseudo-force F_r résultant de la combinaison de ces deux pseudo-forces (figure 3.4-1) est créée (GET_FORCE). Elle traduit l'action de l'utilisateur et celle de l'algorithme à la fois sur le développement du graphe et assure ainsi une réelle interaction entre l'opérateur et l'algorithme, interaction perceptible par le biais de la méthode d'échantillonnage. En étudiant les variations de cette force résultante, on peut ainsi retrouver les trois modes de fonctionnement décrits dans la section 3.2.2.1. En effet, lorsque l'utilisateur déplacera le périphérique rapidement, la pseudo-force F_u deviendra beaucoup plus grande que F_a, la pseudo-force résultante F_r sera très peu différente de F_u et l'opérateur aura donc une influence très largement dominante sur le développement du graphe. Si au contraire l'opérateur bouge lentement l'avatar ou arrête le mouvement, F_r sera très proche de F_a et l'algorithme aura donc une influence beaucoup plus forte sur le développement.

La pseudo-force F_r est calculée comme une combinaison linéaire de F_u et F_a :

$F_r = \alpha.F_u + (1-\alpha).F_a$

Le paramètre α permet de modifier l'influence de l'intention de l'opérateur par rapport à l'algorithme. Si α est initialisé à zéro, l'algorithme équivaut à un RRT classique. S'il est

initialisé à 1, l'algorithme effectuera un simple suivi de cible, la cible étant la position de l'avatar. Les deux pseudo-forces que nous venons de calculer sont donc les données en entrée de notre algorithme. Elles seront donc utilisées dans les étapes qui suivront.

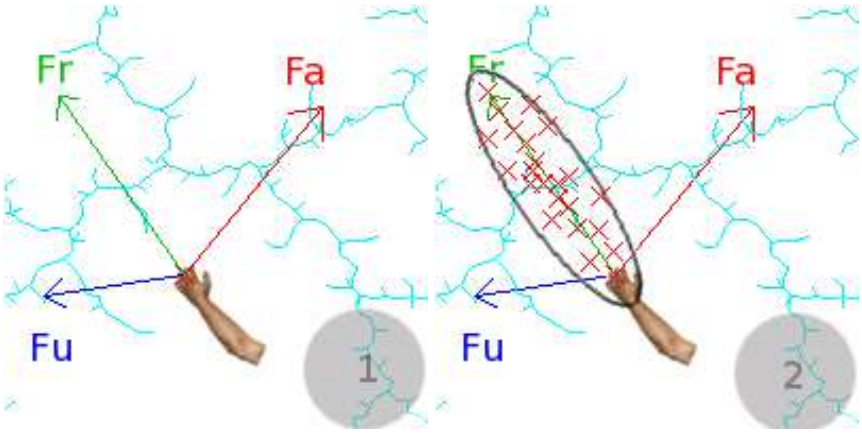


FIG. 3.4 – Calcul de la pseudo-force résultante F_r et échantillonnage.

La méthode d'échantillonnage

Dans les paragraphes précédents, nous avons parlé du calcul de plusieurs pseudo-forces, qui sont destinées à être utilisées lors de l'étape d'échantillonnage. Dans beaucoup de méthodes de planification probabilistes, l'échantillonnage est crucial car c'est lui qui permet de déterminer de quelle façon va se développer le graphe. Permettre à l'utilisateur de contrôler l'échantillonnage, c'est lui permettre de diriger la recherche de chemin. Il existe de nombreuses méthodes d'échantillonnage [Hastings 1970; Wilmarth et al. 1999a; Barraquand et al. 1997; Kuffner 2004] adaptées à différents types de problèmes. C'est sur l'échantillonnage que repose l'interactivité entre l'opérateur et l'algorithme.

Dans la méthode de planification interactive, nous avons implémenté une méthode spécifique d'échantillonnage qui permet de prendre en compte les actions de l'utilisateur sur le périphérique afin d'influencer la recherche. Cette méthode est basée sur un échantillonnage gaussien de l'espace des configurations. La pseudo-force F_r précédemment calculée permet de définir une zone d'échantillonnage gaussienne elliptique centrée autour de la position de l'opérateur et dirigée selon sa direction. L'échantillonnage gaussien est effectué séparément pour chaque degré de liberté (*SAMPLED_CONFIG*). La déformation de la zone d'échantillonnage dans la direction de F_r se fait en augmentant l'écart type de la gaussienne sur l'axe porté par cette direction. Cette déformation le long de l'axe du mouvement de l'utilisateur permet une certaine anticipation de ses mouvements et permet ainsi à l'algorithme de se développer en avance sur la position de l'avatar. Les n dimensions de l'espace des configurations sont donc échantillonnées séparément avec une probabilité gaussienne centrée autour de la position de l'opérateur et un

écart type égal pour chaque dimension, sauf pour la dimension colinéaire à la direction de la pseudo-force F_r. Pour cette dimension, l'écart type est proportionnel à la pseudo-force.

Si F_r n'est colinéaire à aucune des dimensions de l'espace des configurations, un changement de base est effectué en utilisant le procédé de gram-schmidt (*GRAM_SCHMIDT_PROCESS*) vers une base qui vérifie cette colinéarité. La zone d'échantillonnage créée est schématisée figure 3.4-2. Cette méthode permet à l'algorithme d'être guidé vers la position de l'utilisateur tout en restant à proximité du graphe déjà construit. La proximité entre le graphe et l'utilisateur facilite la connexion des nouvelles configurations au fur et à mesure de leur création.

En effet, il peut arriver que l'utilisateur ne soit plus à portée du graphe et donc que la connexion à l'arbre initial ne puisse pas se faire. Dans le cas d'un RRT classique avec un seul arbre de diffusion, les déplacements de l'opérateur ultérieurs à ce « décrochage » seraient perdus. Un décrochage se produit lorsque l'algorithme ne parvient plus à développer des nœuds autour de la position de l'utilisateur, généralement à cause d'un obstacle séparant les nœuds les plus proches dans le graphe et la position courante de l'avatar. Pour pallier ce problème, deux solutions ont été envisagées. La première consiste à introduire la création d'arbres locaux dans l'algorithme. Un échantillon non connectable au graphe initial deviendrait donc une nouvelle racine de diffusion, ce qui permettrait de ne pas perdre les déplacements de l'utilisateur intervenant après un décrochage. La seconde consiste à mémoriser le chemin déjà parcouru par l'opérateur. Dans ce cas, l'échantillonnage selon F_r est abandonné au profit d'un échantillonnage autour du chemin mémorisé. L'avantage de cette méthode est de permettre de garder la trace de l'utilisateur même en cas de décrochage (il n'y a qu'un seul arbre de diffusion mais on continue de suivre l'opérateur avec un peu de retard) alors que la solution précédente ne permet pas de se relier à des arbres locaux existants. Nous verrons que suivant les cas, nous appliquerons l'une où l'autre de ces méthodes.

Bien sûr toutes ces méthodes d'échantillonnage peuvent être utilisées conjointement pour améliorer le suivi de l'opérateur et l'exploration de l'espace des configurations. Malgré cela, il subsiste un problème car l'algorithme reste encore trop dépendant de l'utilisateur. En effet la pseudo-force attractive F_a ne permet qu'une exploration locale autour de la position de l'opérateur par l'algorithme. Ce dernier inconvénient est résolu en utilisant une méthode d'échantillonnage plus commune : l'échantillonnage uniforme. La spécificité de cette méthode est qu'elle permet d'avoir une probabilité équivalente pour chaque configurations d'être choisie comme échantillon. Dans notre méthode interactive, l'échantillonnage uniforme est utilisé conjointement avec les autres méthodes. Le ratio d'utilisation entre l'échantillonnage uniforme et les autres types d'échantillonnage est constant et déterminé par un paramètre β (une méthode d'échantillonnage uniforme est utilisée pour avoir un échantillon tout les β échantillons).

L'utilisation d'un échantillonnage uniforme autorise une exploration plus globale de l'espace des configurations par l'algorithme et donc de la scène virtuelle, afin de rassembler davantage de données qui seront par la suite transmises à l'opérateur pour l'aider dans sa recherche de chemin.

Le choix du plus proche voisin

Le choix du plus proche voisin se fait de manière simple en calculant la distance $d = \sqrt{\sum_{i=1}^{n} \alpha_i \cdot (ddl_{i_{sample}} - ddl_{i_{node}})^2}$ séparant la configuration échantillonnée avec chacun des nœuds du graphe. En ce qui concerne le calcul du centre de masse pour l'obtention de la pseudo-force d'attraction, les p plus proches voisins sont calculés de la même manière.

L'extension du graphe

L'étape d'extension du graphe pour la méthode de planification interactive est la même que pour un algorithme de type *RRT*-Connect : le nœud le plus proche de l'échantillon obtenu est étendu en direction de cet échantillon jusqu'à atteindre l'échantillon ou jusqu'à rencontrer un obstacle. Selon le résultat de cette extension, l'algorithme retourne l'un de ces trois états :

- *Collision* : L'extension du graphe n'a pas pu aboutir car le nœud le plus proche est à proximité d'un obstacle. Il n'y a pas de création de nouveaux nœuds.
- *Obstacle* : Le graphe a été étendu partiellement vers la configuration échantillonnée car il y a un obstacle entre celle-ci et le nœud le plus proche.
- *Atteint* : La configuration échantillonnée a été atteinte après l'extension et est donc intégrée dans le graphe.

Dans le cas d'une méthode *RRT* classique, le déroulement d'une itération se termine à cette étape, après avoir ajouté la nouvelle configuration au graphe. Mais dans le cadre de la planification interactive, il nous faut définir maintenant des informations de sortie. Ces informations permettent de fermer la boucle de l'interactivité avec l'opérateur en lui transmettant des "indices" qui peuvent être soit haptiques, soit visuels (ou les deux, en fonction du type de périphérique d'interaction). Dans la suite de cette section nous allons donc décrire les deux étapes restantes pour fermer cette boucle d'interaction.

L'étiquetage des nœuds

Pour transmettre des informations de manière visuelle à l'opérateur, nous avons choisi de le faire sous forme d'une coloration des nœuds du graphe en construction. Un dégradé de couleurs permet de signifier à l'opérateur si la zone représentée par le graphe est favorable ou non à un passage de l'avatar et si cette zone permet de se rapprocher de la configuration finale. Dans un algorithme *RRT* classique, les configurations sont échantillonnées aléatoirement et sont ensuite testées pour savoir si elle sont en collision avec un obstacle de l'environnement. Comme la topologie de la scène n'est pas connue à priori les seules informations que nous pouvons récolter

de l'environnement proviennent de ces tests de collision : l'espace des configurations est-il libre de collisions à la position de l'échantillon ?

Dans l'algorithme interactif, ces tests sont utilisés (voir étape "Extension du graphe") pour étiqueter les nœuds afin d'avoir une information plus globale sur la zone explorée. En fonction de l'état (Collision, Obstacle, Atteint) dans lequel se trouve l'algorithme, des étiquettes sont calculées pour le nœud nouvellement échantillonné et le nœud parent. Deux types d'étiquettes sont calculées pour chaque nœud et prises en compte dans l'algorithme :

- Une étiquette « distance » qui donne une indication sur la distance entre la configuration échantillonnée et la configuration finale. Cela permet d'avoir un guidage global pour l'utilisateur en direction de la configuration finale.
- Une étiquette « collisions » qui donne une indication sur le nombre de collisions résultant des différentes tentatives d'extension pour le nœud considéré (il s'agit du nœud parent du nouvel échantillon). Cette étiquette fournit une information plus locale que la précédente, de façon à éviter les impasses ou les zones trop encombrées.

Les étiquettes de distance sont calculées de la manière suivante pour le nœud nouvellement ajouté au graphe :

- $D_c = ||\vec{q}_c - \vec{q}_{goal}||$ où D_c est la distance entre la nouvelle configuration q_c et la configuration finale q_{goal}.

Une étiquette de collision est une fonction de la distance couverte durant l'extension, du nombre de tentatives d'extension et du nombre de collisions qui en ont résulté. Les étiquettes de collision sont calculées de la manière suivante pour le nœud parent nouvellement étendu :

- $L_c = ||\vec{q}_{near} - \vec{q}_{new}||$ où L_c est la longueur de la nouvelle arête (entre le nœud étendu q_{near} et le nouveau nœud q_{new}).
- $CL_e = \frac{L_c}{\max\limits_{x \leq nbE}(L_x)} \times \frac{nbCollisions_e}{nbExtensions_e}$ où CL_e est l'étiquette de collision du nœud étendu q_{near} et nbE le nombre d'arêtes sortantes du nœud étendu.

Pour le nœud qui vient d'être créé, on distingue deux situations pour calculer l'étiquette de collision :

- $CL_n = CL_e \times (1 + \frac{1}{L_c})$ où CL_n est l'étiquette de collision du nouveau nœud, dans le cas où la configuration échantillonnée n'a pas été atteinte.
- $CL_n = CL_e \times (\frac{1}{L_c})$ où CL_n est l'étiquette de collision du nouveau nœud, dans le cas où la configuration échantillonnée a été atteinte.

Ces deux étiquettes peuvent être utilisées simultanément ou séparément lors de l'affichage pour colorer le nœud représenté dans la scène virtuelle. Pour cela, l'étiquette choisie est utilisée

comme coefficient multiplicatif pour chaque composante (Rouge, Vert, Bleu) de la couleur de base (au choix de l'opérateur) du graphe. Dans le cas d'une utilisation des deux étiquettes simultanément, le coefficient est calculé à partir de ces deux étiquettes :

– $Coef_{color} = \frac{D_c}{CL_n}$

Malgré cette aide visuelle, il peut arriver que l'opérateur choisisse un chemin - en général un passage ou un couloir étroit - dans lequel il reste bloqué. Un affichage, même en trois dimensions, ne peut pas fournir tous les indices visuels dont l'opérateur aurait besoin pour se sortir de toute situation, sans risquer de surcharger la scène virtuelle. C'est pourquoi une aide supplémentaire est donnée à l'opérateur, sous forme de retour d'effort.

Le retour haptique

Une pseudo-force de retour est introduite dans la boucle d'interaction afin d'avoir un guidage de la recherche de chemin plus intuitif pour l'opérateur et de ne pas remplir la scène d'artefacts pouvant être encombrants pour sa compréhension. Pour calculer cette pseudo-force, de type élastique, nous avons utilisé la pseudo-force attractive F_a calculée précédemment :

$F_s = k \times F_a$ où k est un facteur d'échelle, fixé lors de l'initialisation de l'algorithme.

Dans le cas où le périphérique interactif est un bras haptique, cette pseudo-force est calculée dans la boucle externe qui gère uniquement le retour d'effort du bras. Dans cette boucle, F_s est ajoutée à la pseudo-force F_u pour pouvoir calculer la position et la vitesse de l'avatar dans la scène virtuelle. L'opérateur ressent donc constamment un effort de la part du bras haptique qui va le diriger vers une direction préférentielle calculée par l'algorithme. Si l'algorithme est en retard sur le déplacement de l'opérateur, l'effort aura tendance à ralentir la progression de celui-ci et à le ramener vers les nœuds du graphe, resté en retrait. Si l'algorithme a une certaine avance sur l'opérateur, l'effort aura un effet contraire, lui permettant d'avancer plus rapidement vers la direction préférentielle.

Dans le cas d'une souris 6D, la pseudo-force F_s est calculée lors de la récupération des données du périphérique et appliquée sur le déplacement de l'avatar dans la scène virtuelle. Bien que n'ayant pas de sensation directe sur le retour d'effort, l'opérateur verra le déplacement de son avatar ralenti dans le cas où l'algorithme est en retard sur lui et attiré vers la direction préférentielle calculée par l'algorithme dans le cas inverse.

Dans les deux cas de périphériques interactifs, F_u et F_s sont utilisées pour calculer une force résultante qui sert ensuite pour la vitesse et la situation de l'avatar :

– $F_{hr} = F_u + F_s$ où F_{hr} est la force résultante.
– $S_t = \frac{F_{hr} \times T}{m}$ pour la vitesse au temps t, avec T période utilisée pour la boucle et m masse virtuelle de l'avatar déplacé par l'opérateur.

– $X_t = S_t \times T$ pour la situation au temps t.

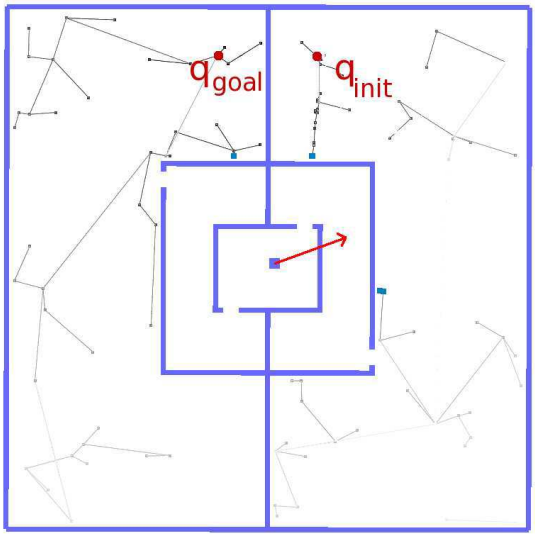


FIG. 3.5 – Retour haptique (flèche rouge) et visuel. Les configurations initiale et finale se situent en haut de la figure, autour du mur central.

Ainsi, avec ces deux retours (le retour visuel et le retour d'effort), la boucle de l'interaction est fermée et permet un échange naturel et continu d'informations entre l'opérateur et l'algorithme pour les problèmes de planification de mouvement. Sur la figure 3.5, on peut voir un exemple simplifié[1] de retour d'effort et de retour visuel. Sur cette exemple, la configuration initiale se situe en haut à droite près du mur central et la configuration finale en haut à gauche du mur central (symétriquement à la configuration initiale). Le chemin solution passe donc par les deux pièces centrales.

Sur la figure on voit que le graphe recouvre déjà les deux parties de part et d'autre des pièces centrales, plus difficiles d'accès à cause des passages étroits. L'utilisateur, représenté par son avatar (la boîte bleue au centre de l'image) est passé sans difficulté dans les pièces centrales et à travers les passages étroits. Mais l'algorithme ayant du retard sur le déplacement de l'utilisateur, il applique une pseudo-force attractive, qui tend à le faire revenir vers le graphe. Les nœuds surlignés en bleu sont les voisins les plus proches de la position de l'opérateur qui ont été utilisés pour le calcul de la pseudo-force. La pseudo-force obtenue est représentée en rouge.

On peut aussi voir la coloration des nœuds, qui se trouvent plus foncés lorsqu'ils sont près de la configuration finale (seules les étiquettes de distance sont utilisées ici) et plus clairs lorsqu'ils s'en éloignent.

[1]Cette exemple a été construit de toutes pièces pour illustrer les principes des retours visuels et haptiques

3.2.3 Expérimentations

Cet algorithme a été implémenté en C++ sur la base de la plateforme HPP (Humanoïd Path Planner) développée au LAAS et basée elle-même sur KineoWorks. Les expérimentations ont été faites sur un PC Dual Core, 2,1 GHz avec 2Go de RAM.

En pratique, l'utilisation d'un échantillonnage uniforme en même temps qu'une méthode introduisant les arbres locaux minimise grandement l'influence de l'opérateur sur la recherche. L'idée de la méthode de planification interactive étant de faire interagir l'opérateur et l'algorithme de manière équilibrée, ces deux composantes ne seront pas utilisées ensemble en pratique. Dans les tests réalisés, l'échantillonnage uniforme sera utilisé avec l'échantillonnage selon la pseudo-force F_r et l'échantillonnage autour du chemin parcouru par l'utilisateur. Dans chacun des cas, le ratio β entre les deux types d'échantillonnage est de 3. Les pseudo-forces (F_u et F_a) utilisées pour calculer la zone d'échantillonnage interactif ont une influence équivalente ($\alpha = \frac{1}{2}$). La détection de collisions est effectuée par le moteur de KineoWorks, nommé KCD. La recherche du plus proche voisin pour l'extension se fait en parcourant tous les nœuds du graphe. Le nombre de nœuds p utilisés comme plus proches voisins pour calculer F_a est égal à 5. Le calcul des plus proches voisins se fait en parcourant tous les nœuds du graphe, mais les p plus proches voisins doivent tous appartenir à la même composante connexe.

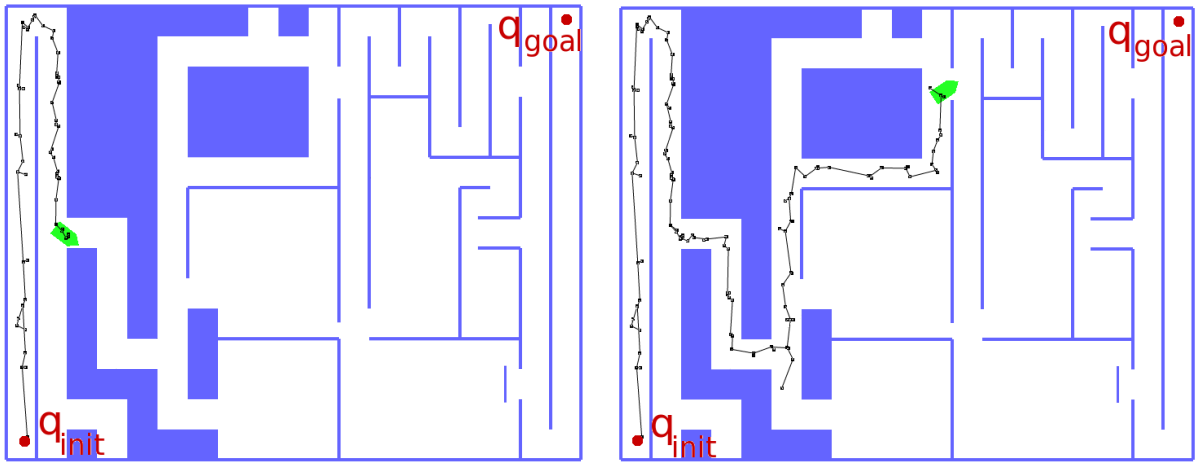
Dans tous les cas, le temps nécessaire à la résolution de la planification est fortement dépendant de l'opérateur et de sa capacité à manipuler le périphérique interactif. Si l'opérateur ne déplace pas le périphérique, ou est bloqué dans une impasse, le temps de planification peut devenir supérieur à celui d'un *RRT* classique car le *IRRT* ne profite pas du guidage de l'utilisateur et se comporte comme un *RRT* classique, à ceci près qu'il continue à échantillonner autour de la position de l'avatar. Dans les différents cas envisagés, l'avatar peut traverser les obstacles et les forces de contact ne sont pas appliquées au bras haptique, de manière à ne pas limiter les déplacements de l'opérateur et à ne pas le bloquer dans des passages difficiles. La force élastique F_s joue le rôle du retour d'effort qui va être ressenti par l'utilisateur. Dans des environnements relativement complexes et sans immersion dans un environnement de simulation 3D, l'opérateur peut être « pris au piège » si les forces de contact sont activées, sans savoir comment sortir de la zone. La désactivation des contacts permet d'éviter ce type de situation et de faciliter le déplacement de l'opérateur, laissant à la partie algorithmique le soin de contourner correctement des obstacles et d'éviter les collisions. Ainsi, même si l'opérateur suit un chemin approximatif qui peut-être en collision sur plusieurs parties, l'algorithme interactif trouvera toujours une solution libre de collisions.

3.2.4 Résultats

La méthode de planification de mouvement interactive que nous avons décrite précédemment a été testée sur différents exemples d'environnement. Le premier environnement permet de mieux appréhender le principe de la méthode et d'illustrer les principaux cas de difficultés rencontrés. Les environnements suivants permettent de voir le comportement de l'algorithme dans des espaces des configurations de dimension plus élevée, avec une application industrielle.

3.2.4.1 Un cas simple

Le but de cet exemple est uniquement d'illustrer le principe de la planification de mouvement interactive et ses différents cas de fonctionnement. Ici, l'utilisateur se sert d'une souris 6D pour déplacer son avatar dans la scène virtuelle représentée sur les figures 3.6 et 3.7. La coloration des nœuds n'a pas été activée sur les figures montrées en exemple.

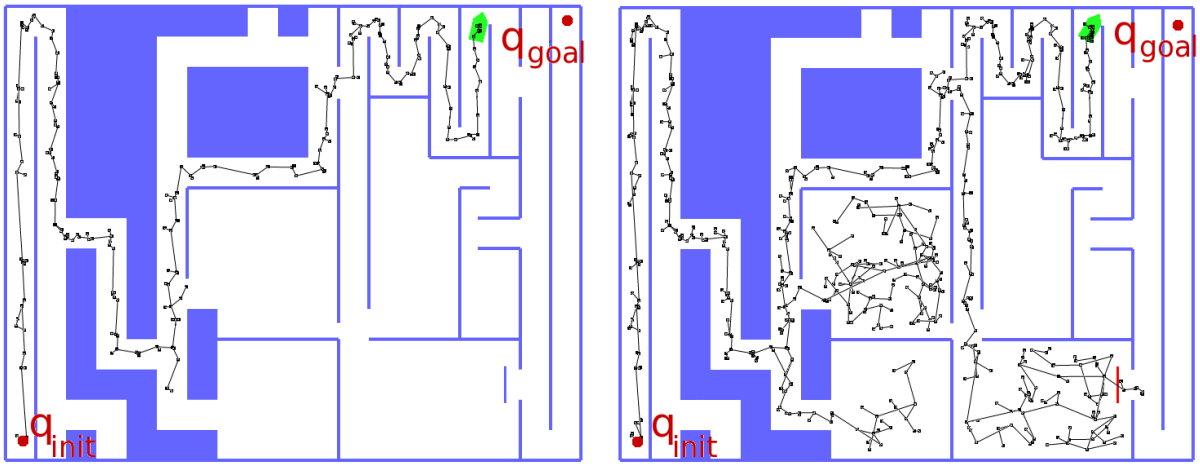


(a) Peu de branches sont développées. (b) L'algorithme suit l'opérateur.

FIG. 3.6 – L'utilisateur a une influence largement dominante.

Le but est de déplacer le bloc vert en forme de « flèche » dans un labyrinthe de la configuration initiale qui se trouve dans le coin inférieur gauche des images, vers la configuration finale dans le coin supérieur droit. Le bloc à déplacer (l'avatar) possède trois degrés de liberté : x, y et θ (l'espace des configurations est de dimension 3).

Au début du labyrinthe, l'utilisateur et l'algorithme peuvent tout deux trouver le chemin à suivre assez facilement car il n'y a qu'une seule direction possible. Le couloir n'étant pas très large, l'utilisateur prend tout de même un peu d'avance, mais l'algorithme est aussi capable de trouver facilement ce chemin.



(a) L'opérateur arrive dans un cul-de-sac. (b) Extension du graphe.

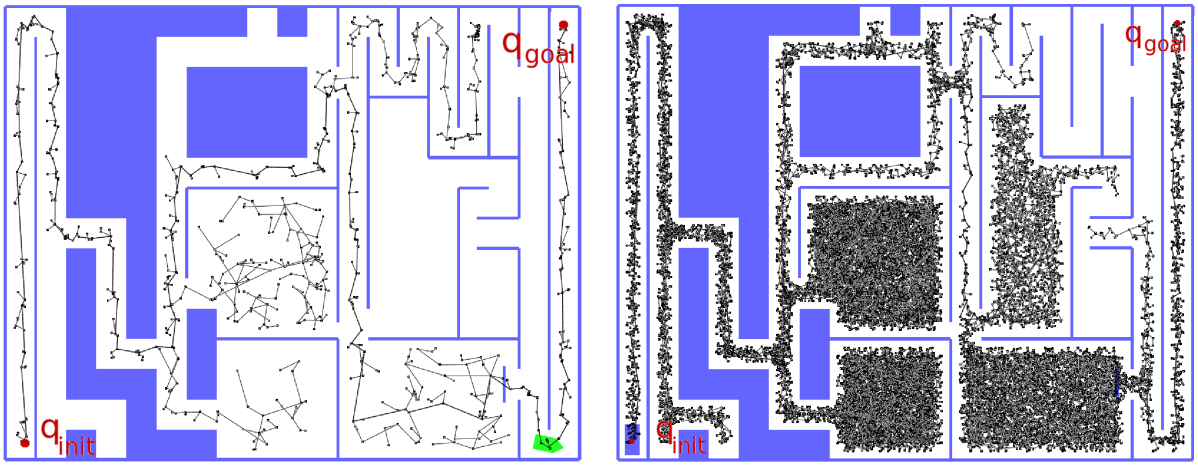
FIG. 3.7 – Sortir du passage.

Ensuite, le chemin solution emprunte un passage plus étroit. Le passage à emprunter est évident pour l'opérateur, mais dans un cas non interactif, l'algorithme explorerait la totalité de l'espace des configurations. De plus, un algorithme *RRT* classique aurait beaucoup de difficultés à trouver l'entrée du passage. L'opérateur avance donc naturellement à l'intérieur du passage et contraint l'algorithme à se développer dans cette direction (figure 3.6-a). L'algorithme effectue donc un simple suivi de l'utilisateur.

Après le passage étroit, on peut voir sur les figures deux pièces sans issues. Ces pièces sont assez larges et nécessite beaucoup de temps de calcul pour être explorées avec un algorithme *RRT* non interactif. L'utilisateur a la possibilité d'éviter ces zones et d'entrainer le développement de l'arbre avec lui afin d'éviter un gaspillage de temps (figure 3.6-b). On peut tout de même voir sur cette figure que l'algorithme initie une exploration indépendamment de l'opérateur : il développe une branche en direction d'une des chambres sans issues.

Sur la figure 3.7-a, l'exploration indépendante de l'algorithme est un peu plus visible (quelques nœuds ont été ajoutés sur le graphe développé à partir du suivi de l'opérateur). L'opérateur a aperçu une solution possible et se dirige donc dans cette direction pour l'essayer. Mais le passage devient trop étroit pour que l'avatar puisse tourner et l'utilisateur reste donc « bloqué » à cet endroit. En réalité, il n'est pas bloqué mais dans la pratique il passe du temps à laisser se développer le graphe dans cette zone pour qu'il continue à travers ce passage. De plus, la pseudo-force d'attraction contraint l'opérateur à rester proche du graphe, il ne peut donc pas trop s'éloigner de sa position sur l'image 3.7-a s'il souhaite passer par ce chemin afin de laisser à l'algorithme le temps de se développer dans cette région.

Durant le temps où l'opérateur essaie de sortir du passage étroit[2], l'algorithme continue à explorer l'environnement indépendamment de la position de l'avatar. Sur la figure 3.7-b, les pièces sans issues ont été explorées et l'algorithme trouve un autre passage que l'opérateur n'a pas choisi car il semble trop étroit pour l'avatar. Cette erreur de l'opérateur est due à son point de vue sur la scène virtuelle, en réalité, l'obstacle (en rouge sur la figure) que l'algorithme semble traverser se situe en dessous du plan sur lequel l'opérateur déplace son avatar. Ce « piège » peu sembler artificiel, mais il illustre parfaitement les problèmes qu'un opérateur peut rencontrer dans des environnements plus complexes, à savoir des chemins solutions dont la représentation est peu évidente ou peu intuitive et des passages qui semblent être des solutions évidentes mais ne peuvent pas être traversés. Ces problèmes seront aussi illustrés dans le cas d'un environnement industriel (section 3.2.4.3).



(a) Le graphe de la méthode interactive.

(b) Le graphe d'un RRT classique.

FIG. 3.8 – Comparaison d'un graphe de RRT interactif et de RRT Classique.

Finalement, l'utilisateur s'aperçoit que le graphe s'est développé par ailleurs et offre une meilleure solution, il déplace alors son avatar jusqu'à la partie du graphe plus avancée. Il continue ensuite à guider la recherche jusqu'à la configuration finale. La figure 3.8 montre le graphe construit par la planification interactive après que la solution ait été trouvée et le même graphe construit par un RRT classique. Là où l'interactivité avec l'opérateur a permis une exploration rapide (quelques minutes), le RRT a effectué une recherche dense et a couvert le maximum d'espace. Ce type d'exploration consomme beaucoup de temps et de ressources ($>$ 2 heures pour le RRT).

[2]Les différentes figures n'ont pas été prises avec un intervalle de temps régulier : les trois premières sont prises à quelques secondes d'intervalle, tandis que la dernière est prise plusieurs dizaines de secondes après

3.2.4.2 Avec des degrés de liberté supplémentaires

Dans cette exemple représenté figure 3.9, l'opérateur utilise le bras haptique « Virtuose6D » pour déplacer un objet en forme de S afin de traverser un mur avec une ouverture. Le mur fait 700x700x20 et l'ouverture de 80x80 est placé au centre du mur. L'objet est formé de trois « bâtons » de 20x20x100. L'ouverture est trop petite pour laisser passer l'objet en ligne droite. Pour trouver le chemin solution, il faut que l'objet opère plusieurs rotations complexes autour de différents axes afin de faire passer chaque bâton dans l'ouverture. L'environnement ne présente qu'un seul passage difficile et c'est à cet endroit que l'aide de l'opérateur est intéressante. Pour faciliter sa tâche, nous avons décidé d'autoriser la création d'arbres locaux. Tout le travail de guidage se fait donc au niveau de l'ouverture.

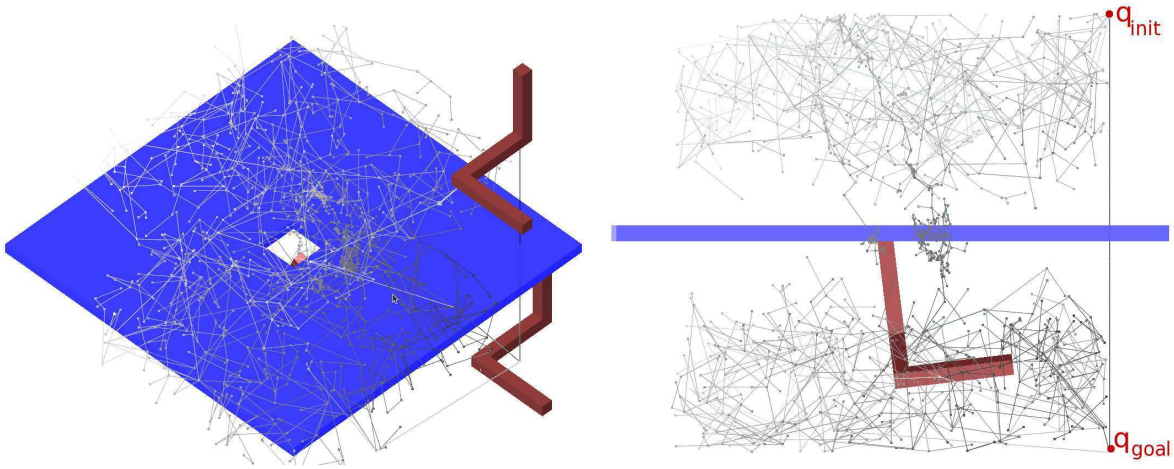


FIG. 3.9 – Un problème avec un espace des configurations de dimension 6.

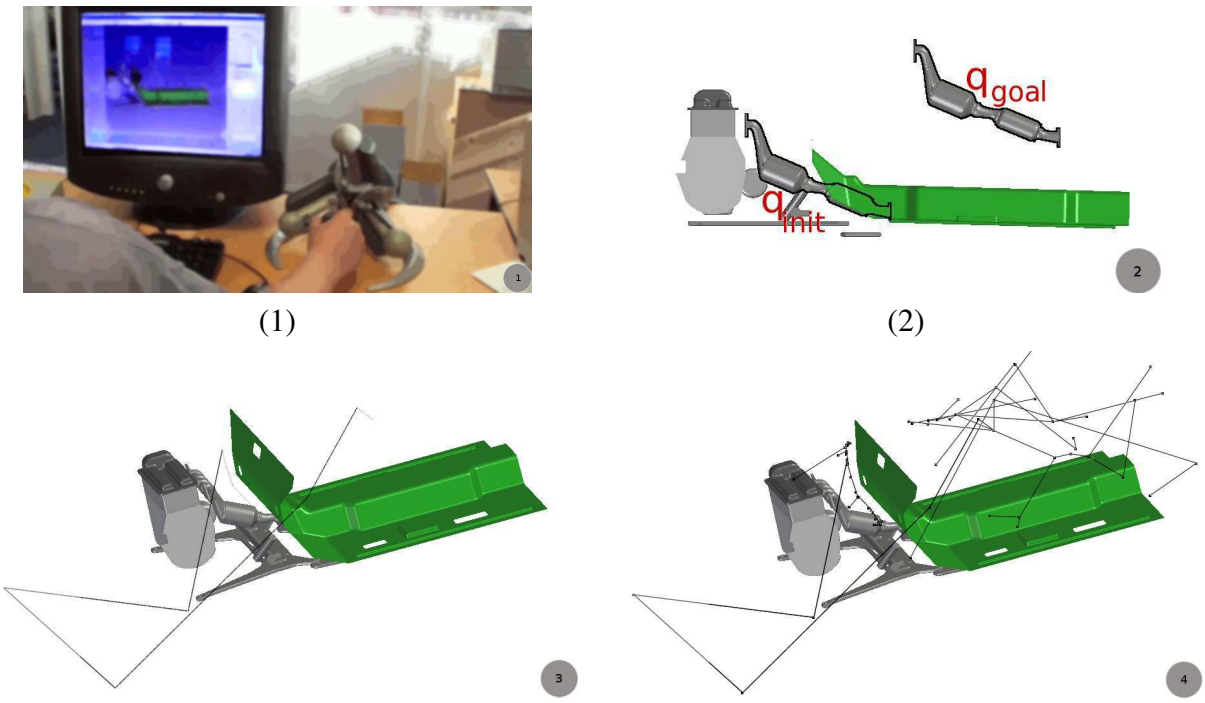
Les *ddl* de l'objet sont bornés afin de l'obliger à passer par l'ouverture, la configuration initiale se situant au dessus du mur et la configuration finale en dessous. Les nœuds du graphe représentent la position du centre du cube dessiné par l'objet (position qui n'est pas sur l'objet lui-même). C'est pour cette raison que sur la figure 3.9 le graphe semble passer à travers le mur. On peut aussi voir la coloration des nœuds : plus on est éloigné de la configuration finale, plus la couleur du nœud est claire. On remarque sur l'image de droite que la partie du graphe qui passe à travers l'ouverture est plus foncée, indiquant que l'algorithme a donné la bonne direction préférentielle en calculant les étiquettes.

La résolution de cette exemple nécessite une certaine dextérité avec le périphérique interactif de la part de l'utilisateur car le passage est assez étroit. Même si les forces de contact sont désactivées, il faut que l'opérateur guide le développement de l'arbre à l'intérieur du trou et donc ne s'écarte pas trop de la zone sans collisions (ce que la pseudo-force attractive calculée à partir du graphe aide à faire). La recherche de solution dépend de la dextérité de l'utilisateur.

Un opérateur expérimenté pourra trouver la solution beaucoup plus rapidement qu'un néophyte. On voit sur la figure 3.9 que le graphe s'est développé de part et d'autre du mur et a couvert une grande partie de l'espace des configurations : cette couverture n'a pas été faite par le guidage de l'utilisateur mais par la partie indépendante de l'algorithme. La densité de cette partie du graphe peut témoigner de l'expérience de l'opérateur : plus l'opérateur met du temps à traverser l'ouverture, plus le graphe peut se développer ailleurs. Ici, le graphe développé n'est pas très dense, ce qui montre une dextérité moyenne de l'utilisateur.

3.2.4.3 Un cas industriel

L'exemple suivant est basé sur un cas réel d'utilisation emprunté à l'industrie automobile. Le but de cet exemple est de trouver un chemin sans collisions permettant de désassembler une partie d'une voiture (un pot d'échappement) pour vérifier si le processus de fabrication est correct ou si la maintenance de la voiture sera possible ou même aisée après sa sortie d'usine.



(1) (2)

(3) (4)

FIG. 3.10 – Un cas industriel.

Sur cette exemple, l'espace des configurations est borné pour simuler les parties manquantes de la voiture qui pourraient modifier le chemin final. Les données en entrée sont le modèle CAO d'un pot d'échappement et de son logement et les configurations initiale et finale pour le pot d'échappement (figure 3.10-2). Le but de l'utilisateur est de retirer le pot d'échappement par le

haut. L'opérateur manipule un bras haptique (figure 3.10-1) pour déplacer le pot dans le modèle CAO, avec un retour d'effort calculé par l'algorithme (force attractive). Le développement à partir de la configuration finale a été autorisé dans cet exemple car il modifie peu l'interaction algorithme-utilisateur : la configuration finale se trouve dans un espace libre, facilement accessible après l'extraction du pot d'échappement.

Le problème est résolu en quelques minutes par l'utilisateur (figure 3.10-3). La difficulté majeure consiste à sortir le pot de son emplacement sous la partie verte, à droite. Ce problème de désassemblage peut être résolu plus rapidement par des algorithmes spécifiques ([Ferré and Laumond 2004]) qui sont entièrement automatiques. Comme tout problème de désassemblage, il s'agit de retirer un objet d'une zone étroite vers une zone libre. Bien que ce type de problème ne favorise pas l'interaction entre algorithme et utilisateur, il est important de noter que dans le cadre du guidage de l'utilisateur cette méthode reste valide et performante car l'interactivité permet de contrôler la solution, notament par rapport aux critères métier. Sur la figure 3.10-4, on peut voir le graphe final après la résolution du problème. On peut remarquer que le graphe s'est développé surtout autour des positions initiale et finale. Pour la première partie, le développement s'est fait surtout sous l'influence de l'utilisateur. Pour la partie autour de la position finale, le développement est dû à la partie indépendante de l'algorithme. En observant la partie du graphe autour de la position initiale, il est possible de voir qu'une fois le pot extrait du passage étroit, la connexion avec l'autre partie du graphe se fait très facilement (un minimum de nœuds de transition et de longues arêtes).

3.2.4.4 Analyse

Du fait de son principe de base, l'interaction avec un utilisateur humain, cette méthode est fortement dépendante des capacités de l'opérateur en question : sa capacité à manipuler le périphérique interactif avec plus ou moins de précision, à percevoir les indices fournis par l'algorithme, à visualiser la scène et éventuellement les passages difficiles et le chemin. Là où un utilisateur avancé guiderait efficacement la recherche et trouverait rapidement la solution, un opérateur inexpérimenté pourrait ralentir la résolution du problème par rapport aux méthodes probabilistes classiques. Au-delà de la dextérité de l'opérateur, il y a plusieurs paramètres pouvant modifier la vitesse de résolution à prendre en compte.

Un facteur déterminant pour la rapidité de résolution et lié à cette dextérité est la sensibilité attribuée au périphérique interactif. En fonction de la taille de l'environnement et des passages étroits, une plus grande sensibilité (pour un même effort de l'utilisateur, le déplacement dans la scène virtuelle est plus grand) peut améliorer la précision du guidage mais aussi le rendre plus difficile en exacerbant les défauts de mouvement de l'opérateur. De plus, l'écart type de l'échantillonnage gaussien autour de la position de l'utilisateur joue aussi un rôle important. Si

l'écart type est trop important, le guidage par l'opérateur aura une efficacité moindre dans les passages étroits. D'un autre côté, s'il est trop faible, le guidage par l'algorithme sera minimisé. Le facteur k utilisé pour calculer le retour d'effort F_s a aussi une grande influence sur les mouvements de l'utilisateur. En effet si k est grand, l'opérateur aura une forte contrainte lui imposant de rester très près du graphe. Dans un environnement encombré ou étroit, cela permet de garder un distance raisonnable entre le graphe et l'opérateur, afin de toujours pouvoir suivre les mouvements de ce dernier. Dans un environnement plus spacieux, un k plus faible permet de laisser plus de liberté à l'opérateur pour diriger la recherche.

Le choix de la méthode d'échantillonnage a aussi une forte influence sur l'efficacité de l'interaction (outre son influence sur la partie automatique de la planification). Les méthodes basées sur le suivi de l'utilisateur uniquement (échantillonnage autour de la position courante et échantillonnage autour du chemin parcouru) n'apporteront qu'un avantage local pour la recherche de chemin (évitement d'obstacles et correction d'un chemin de l'utilisateur éventuellement en collision), mais permettront de focaliser le développement du graphe sur celle-ci et ainsi d'éviter une exploration globale plus coûteuse en temps et en calculs. Les méthodes d'échantillonnage mixtes (incluant donc une partie uniforme sur l'espace des configurations) pourront cumuler les avantages locaux avec les avantages d'une exploration plus globale de la scène virtuelle (acquisition d'informations supplémentaires, pouvoir trouver des solutions non-évidentes pour un humain). Ces dernières méthodes réduisent le contrôle de l'utilisateur sur le développement du graphe. De la même façon, si on compare les méthodes basées sur le suivi de la position à celles basées sur le suivi du chemin, on remarque que les premières offrent plus de contrôle à l'opérateur que les secondes. En effet, lors du suivi de position, le développement du graphe se concentre sur la position courante de l'utilisateur. Ceci offre de meilleures possibilités de contrôle à l'opérateur mais peut amener l'algorithme à décrocher (le développement du graphe se poursuivra, mais pas forcément selon l'intention première de l'utilisateur, sauf si celui-ci revient sur ses pas). Lors d'un suivi de chemin, ce phénomène ne se produit pas, mais dans une situation ou un suivi de position décrocherait, le suivi de chemin introduirait un retard par rapport à l'utilisateur, ce qui aurait pour conséquence une aide très réduite à l'opérateur par l'algorithme durant la période où le retard n'est pas rattrapé.

Enfin, le dernier facteur à prendre en compte détermine directement la part d'influence de chaque partie de la coopération. Il s'agit du paramètre α utilisé dans le calcul de la pseudo-force résultante F_r dont la méthode d'échantillonnage par suivi de position se sert pour effectuer l'échantillonnage.

En résumé, tous ces paramètres peuvent être modifiés en fonction du but et du contexte de la recherche de chemin. Si le chemin final doit avoir un tracé plus naturel ou plus intuitif pour

un humain (ce qui était l'un des objectifs du projet AMSI), l'accent sera mis sur le contrôle par l'opérateur du développement du graphe. Si au contraire c'est la résolution qui importe le plus et non l'aspect « humain », une influence plus importante pourra être accordée à l'algorithme, à travers ces différents paramètres.

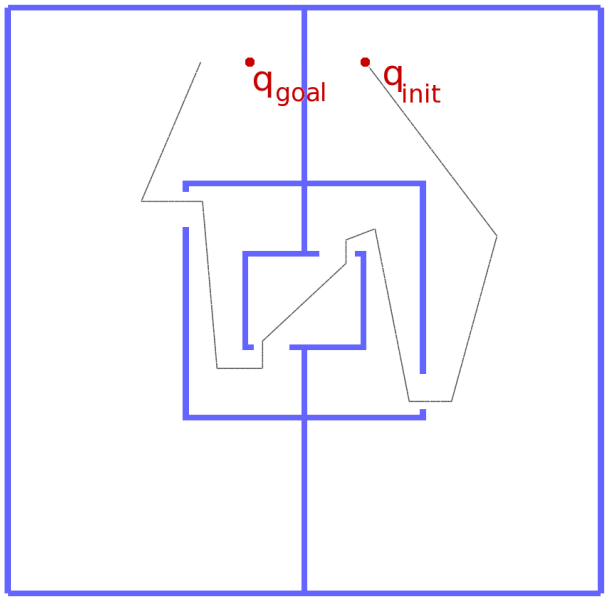


FIG. 3.11 – Environnement utilisé pour les tests. Le chemin de l'utilisateur est représenté en noir et le chemin direct entre la configuration initiale et la configuration finale est représenté en vert. Le robot est représenté à la configuration initiale.

Parmi tout ces paramètres, il nous a semblé plus important de nous concentrer sur l'influence de la dextérité de l'utilisateur, celle de la méthode d'échantillonnage ainsi que l'écart type pour chacune de ces méthodes. Pour les besoins des tests nous avons choisi un environnement simple : celui de la double chambre illustré en figure 3.11. La dextérité de l'utilisateur est un paramètre difficile à modéliser. Nous l'avons représentée par la vitesse de déplacement de l'utilisateur. Les mouvements de l'utilisateur sont donc simulés par un chemin prédéfini noté $traj_p$. La chemin de l'opérateur est parcouru sans qu'il y ait de contact avec les obstacles (pas de chemin approximatif). À chaque itération (à chaque fois qu'une nouvelle configuration est échantillonnée) de l'algorithme, la vitesse de l'utilisateur est simulée en avançant d'un pas ε_v le long de ce chemin. ε_v sera utilisé comme indicateur de la dextérité de celui-ci. Nous avons testé quatre méthodes d'échantillonnage :

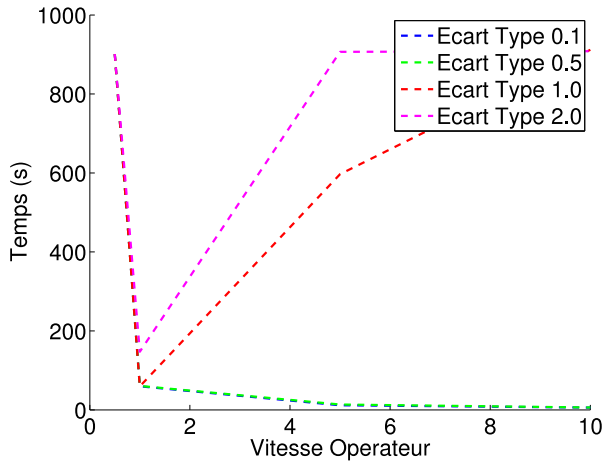
– Échantillonnage gaussien autour de la position utilisateur. La méthode sera nommée *MAG*.
– Échantillonnage gaussien autour du chemin parcouru de l'utilisateur. La méthode sera nommée *PATH*. Il ne s'agit pas du chemin $traj_p$ complet, mais de la portion qui a déjà

été parcourue le long de $traj_p$ depuis le début de l'exécution. Il est nommé $traj_c$ et est modifié à chaque itération, après le mouvement simulé de l'opérateur.
– Échantillonnage gaussien autour de la position utilisateur et avec un échantillonnage uniforme en plus. La méthode sera nommée $MAG+CS$. Dans la pratique, l'ajout d'une composante d'échantillonnage uniforme se fait en échantillonnant de manière uniforme une fois toutes les β itérations. Ici $\beta = 3$.
– Échantillonnage gaussien autour du chemin parcouru de l'utilisateur $traj_c$ avec un échantillonnage uniforme en plus. La méthode sera nommée $PATH+CS$.

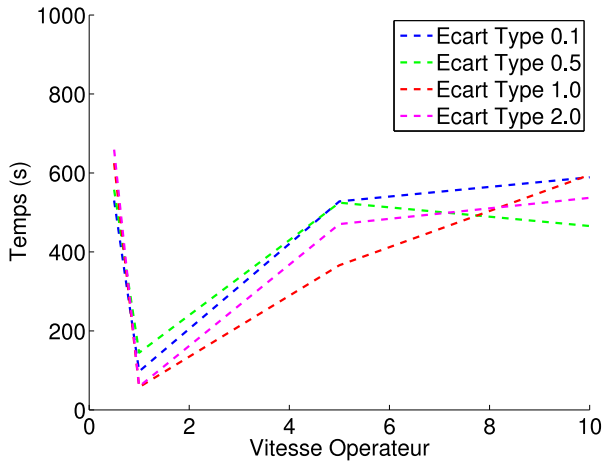
Chaque méthode étant basée sur un échantillonnage gaussien, elles sont donc affectées par la valeur d'écart type. Nous avons testé chaque méthode pour quatre écarts type différents et pour quatre ε_v différents. Les retours visuels et haptiques ne seront pas considérés ici. Tous les résultats présentés ici sont des moyennes sur 50 tests. La durée de la recherche de chemin a été limitée à 900 secondes pour chaque test.

La figure 3.12 permet de comparer le temps de résolution en fonction de la vitesse de l'opérateur ε_v pour chaque valeur d'écart type. Chaque courbe (de (a) à (d)) présente les résultats pour une méthode d'échantillonnage particulière. La courbe (a) concerne un échantillonnage uniquement autour de la position courante de l'utilisateur (méthode MAG) . On remarque que pour des valeurs d'écart type faibles (tracés bleu et vert), l'algorithme profite pleinement du guidage de l'utilisateur et donc de sa dextérité, étant donné que le temps de résolution diminue avec la vitesse de l'opérateur. L'algorithme est peu dépendant de l'écart type lorsqu'il n'y a pas de décrochage. D'un autre côté, lorsque l'écart type est plus grand (tracés rouge et rose), l'algorithme se démarque un peu plus du guidage de l'utilisateur, pour explorer les alentours de sa position. Mais si ce dernier est trop rapide, l'algorithme ne parvient plus à suivre sa position correctement, d'où un décrochage et donc une augmentation du temps de résolution. A partir d'une vitesse trop grande, l'algorithme sera même incapable de trouver une solution étant donné la méthode d'échantillonnage (sur le tracé rose, on peut voir que l'algorithme atteint la limite de temps, à 900 secondes, à partir d'un ε_v égal à 5 ; pour le tracé rouge, c'est à ε_v égal à 10).

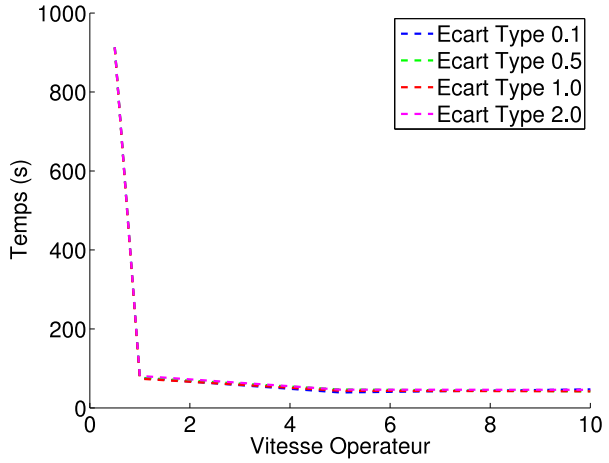
La courbe (b) concerne la même méthode d'échantillonnage, mais avec une composante uniforme sur CS en plus. On remarque que l'ajout de cette composante introduit un retard sur le temps de résolution, quel que soit la valeur de l'écart type. Ce retard augmente avec la vitesse de l'opérateur, ce qui fait que l'algorithme ne peut pas profiter de sa dextérité. Mais la composante uniforme permet aussi de pallier ce retard et de compléter la résolution dans le temps imparti quel que soit l'écart type. De la diminution de la pente des tracés, on peut déduire que l'algorithme est de moins en moins affecté par la vitesse de l'utilisateur, étant donné que la composante uniforme devient le seul vecteur de propagation de l'arbre au fur et à mesure que la vitesse augmente.



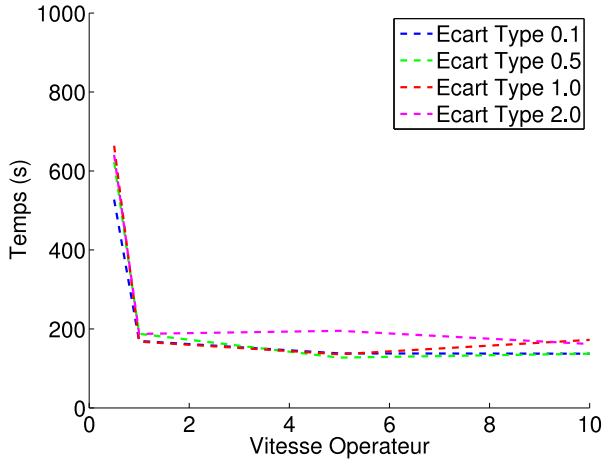
a - Autour de la position opérateur (MAG)



b - Autour de la position opérateur et uniforme ($MAG+CS$)



c - Autour du chemin opérateur ($PATH$)



d - Autour du chemin opérateur et uniforme ($PATH+CS$)

FIG. 3.12 – Comparaison des effets de l'écart type sur le temps de résolution en fonction de la vitesse de l'opérateur pour différentes méthodes d'échantillonnage.

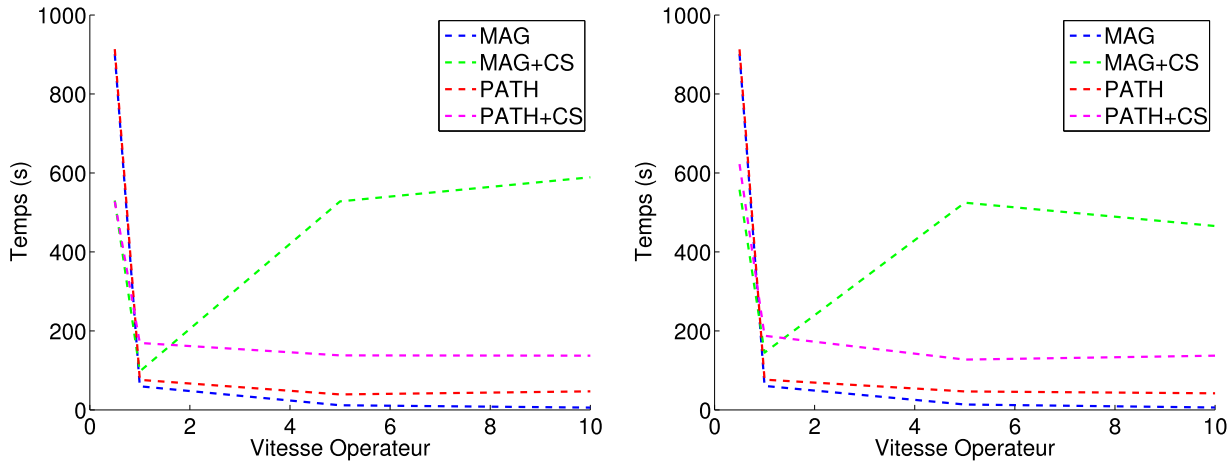
La courbe (c) montre les résultats pour un échantillonnage gaussien autour du chemin $traj_c$ de l'utilisateur. On voit clairement sur cette courbe que l'algorithme tire avantageusement parti de la vitesse de l'opérateur car le temps de résolution diminue avec celle-ci. Il n'y a pas de décrochage possible car le chemin est mémorisé. Cette méthode d'échantillonnage n'est pas affectée par le changement d'écart type pour la même raison.

La courbe (d) concerne le même type d'échantillonnage, mais avec une composante uniforme en plus. On remarque que par rapport à la méthode $PATH$, l'écart type affecte légèrement le temps de résolution qui augmente avec sa valeur. La composante uniforme de l'échantillonnage introduit donc ici aussi un retard sur la résolution. Mais cette composante permet de développer le graphe dans des domaines de CS non explorés par l'utilisateur qui peuvent se révéler utiles en cas de blocage.

De manière plus générale, on peut donc remarquer que l'ajout d'une composante uniforme à une méthode d'échantillonnage ajoute un retard sur la résolution de la recherche de chemin qui peut être plus ou moins conséquent en fonction de la méthode. Les méthodes sans composante uniforme exploitent mieux la dextérité de l'utilisateur, mais peuvent être facilement bloquées en cas de décrochage ou si le chemin $traj_p$ de celui-ci est approximatif (c'est-à-dire parfois en collision). On voit aussi en comparant les différentes courbes que les méthodes basées sur un échantillonnage autour de $traj_c$ sont beaucoup moins affectées par le changement d'écart type que les autres.

Sur les 4 courbes, on voit au début que le temps de résolution est très important : c'est dû au fait que la vitesse de l'opérateur est trop petite pour lui permettre d'atteindre la configuration finale dans le temps imparti.

La figure 3.13 permet de mieux observer les différences entre les méthodes d'échantillonnage au niveau du temps de résolution par rapport à la vitesse de l'opérateur. Chacune des quatre courbes montre le cas pour une valeur d'écart type précise. Sur la courbe (a), pour la plus petite valeur d'écart type, on voit que la méthode d'échantillonnage MAG est la plus efficace en temps de calcul. Elle permet de mieux tirer profit de la vitesse utilisateur. On remarque que les différences de performance entre cette méthode et la méthode $PATH$ sont assez négligeable. Lorsque l'on ajoute une composante uniforme, on observe que le temps de calcul augmente sensiblement (tracés rose $PATH + CS$ et vert $MAG + CS$). On peut tout de même voir que la méthode $PATH + CS$ suit le même type de tracé que les méthodes sans composante uniforme. Elle profite aussi de la dextérité de l'utilisateur. Quand à la méthode $MAG + CS$, l'ajout de la composante uniforme ralentit le suivi de l'utilisateur, c'est pourquoi les performances se dégradent au fur et à mesure que la vitesse augmente. Cela ne se produit pas pour la méthode $PATH + CS$ car la composante $PATH$ de la méthode permet de mémoriser le chemin de l'utilisateur.



a - Pour un écart type de 0.1

b - Pour un écart type de 0.5

c - Pour un écart type de 1.0

d - Pour un écart type de 5.0

FIG. 3.13 – Comparaison des effets de la méthode d'échantillonnage sur le temps de résolution en fonction de la vitesse de l'opérateur pour différents écarts types.

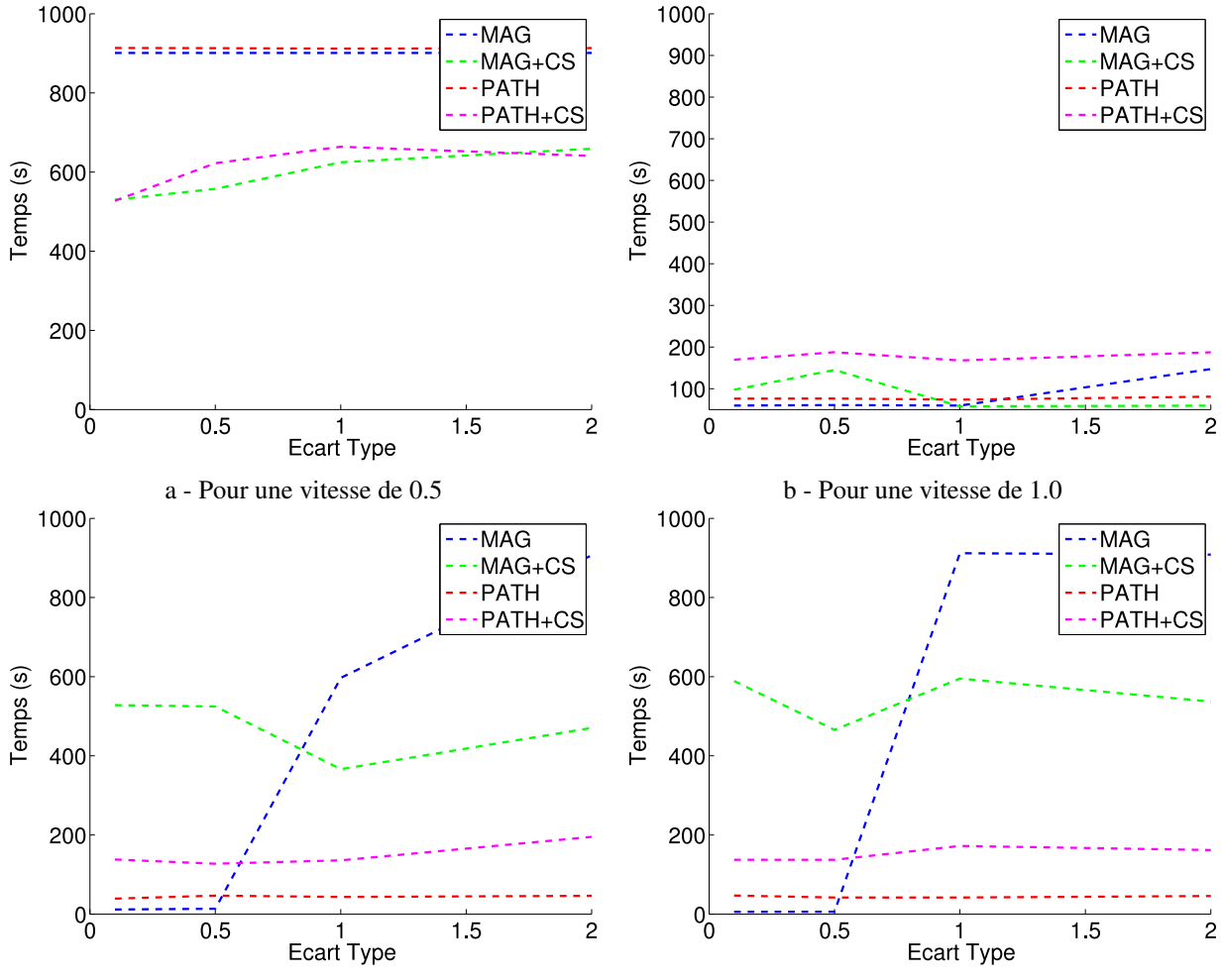
La courbe (b) montre les temps de résolution en fonction de la vitesse de l'utilisateur pour une valeur d'écart type de 0.5. Cette courbe est similaire à la courbe (a), les même remarques s'appliquent donc ici.

La courbe (c) concerne les résultats pour une valeur d'écart type de 1,0. On peut observer que les méthodes basées sur le suivi de la position de l'utilisateur (*MAG* et $MAG + CS$) sont pénalisées par l'augmentation de l'écart type. Le suivi de l'utilisateur est moins précis et l'algorithme profite donc moins de sa dextérité et de son expérience. L'algorithme est donc plus vulnérable aux décrochages, notamment lors de la traversée des passages étroits qui sont présents dans l'environnement considéré. Néanmoins, l'ajout d'une composante uniforme ($MAG + CS$) permet d'atténuer la dégradation des performances.

La courbe (d) confirme les observations faites sur la courbe (c) en accentuant la dégradation des performances des méthodes basées sur *MAG*. On voit tout de même que la composante uniforme de $MAG + CS$ augmente son influence au fur et à mesure que l'utilisateur augmente sa vitesse et stabilise ainsi les performances de l'algorithme. On remarque aussi que la méthode *MAG* ne parvient pas à trouver de solution dès que l'utilisateur a une vitesse supérieure à 5.

De manière générale, on voit que les méthodes $PATH$ et $PATH + CS$ sont plus stables par rapport au changement d'écart type et ont des performances similaires en termes de temps de calcul lorsque l'écart type devient plus grand. L'ajout d'une composante uniforme augmente légèrement le temps de calcul si la méthode d'échantillonnage de base choisie ($PATH$ ou *MAG*) permet de résoudre le problème. Dans ce cas là, cette composante uniforme devient inutile. Dans le cas contraire, elle permet de pallier les défauts de la méthode en échantillonnant globalement dans *CS* et non localement autour d'une position donnée. De même que pour la figure 3.12, les temps de résolution au début des courbes sont importants car la vitesse de l'opérateur est trop faible pour lui permettre d'atteindre la configuration finale dans le temps imparti. On observe tout de même que les méthodes ayant une composante d'échantillonnage uniforme (tracés rose et vert) parviennent à résoudre le problème dans la limite de temps. En comparant sur chaque courbe les valeurs de début et de fin du tracé vert, on voit qu'elles sont proches : les temps de résolution sont similaires, ce qui confirme le fait que c'est la composante uniforme de la méthode d'échantillonnage qui permet de résoudre la planification dans ce cas.

La figure 3.14 présente un point de vue différent des mêmes résultats. Elle permet de comparer les temps de résolution des différentes méthodes d'échantillonnage en fonction de l'écart type. Chaque courbe présente ces résultats pour une vitesse de l'opérateur donnée. Sur la courbe (a), on peut observer les résultats pour une vitesse de 0,5. Comme on a pu le voir sur les figures précédentes, les méthodes n'ayant pas de composante uniforme ne parviennent pas à résoudre le problème avant l'échéance car l'utilisateur n'est pas assez rapide pour atteindre



a - Pour une vitesse de 0.5

b - Pour une vitesse de 1.0

c - Pour une vitesse de 5.0

d - Pour une vitesse de 10.0

FIG. 3.14 – Comparaison des effets de la méthode d'échantillonnage sur le temps de résolution en fonction de l'écart type pour différentes vitesses de l'opérateur.

la configuration finale avant. C'est pourquoi les tracés rouge et bleu ont un temps de résolution constant ayant pour valeur la limite de temps. En ce qui concerne les deux autres méthodes, on voit qu'elles parviennent dans tous les cas à résoudre le problème. Cela est dû à l'ajout d'une composante uniforme aux méthodes d'échantillonnage. Le fait que leurs tracés croissent avec l'écart type est dû au fait que l'utilisateur guide tout de même l'algorithme, même s'il ne parvient pas jusqu'à la configuration finale. La précision du guidage étant influencé par l'écart type, les temps de résolution en sont aussi influencés.

Sur la courbe (b), qui montre les résultats pour une vitesse opérateur de 1, on remarque que tous les temps de résolution sont à peu près stables, quelle que soit la valeur de l'écart type. Seule la méthode *MAG* tire un léger avantage de l'augmentation de l'écart type. Ceci est dû à l'ajout de la composante uniforme, car on peut voir que la méthode *MAG* augmente légèrement lorsque l'écart type augmente, au contraire de $MAG+CS$.

Les courbes (c) et (d) montrent des tracés similaires pour chaque méthode. Les méthodes $PATH$ et $PATH+CS$ restent stables et les courbes montrent que la méthode *MAG* est très peu adaptée pour des écarts types et des vitesses opérateur trop importants. La méthode $MAG+CS$ reste peu affectée par l'écart type. Les légères différences perçues sur les tracés verts sont dues à la composante uniforme. La légère différence entre le tracé de $MAG+CS$ sur la courbe (c) et celui sur la courbe (d) est due au fait que la composante *MAG* garde tout de même une légère influence sur l'algorithme et donc sur le temps de résolution.

En résumé, les figures analysées dans cette section montrent que la méthode d'échantillonnage, la vitesse de l'utilisateur et l'écart type sont 3 paramètres importants qui influent beaucoup sur le temps de résolution. Ils doivent être adaptés en fonction des capacités de l'utilisateur. Les méthodes de suivi simple (sans composante uniforme) sont plus adaptées à un utilisateur expérimenté qui pourra effectuer un déplacement rapide et précis. Dans ce cas, ces méthodes permettront de tirer pleinement avantage du guidage de l'utilisateur. Les méthodes comportant une composante uniforme donnent de meilleurs résultats si l'utilisateur est inexpérimenté ou a des difficultés à trouver la solution. Elles permettent d'avoir une exploration globale de l'environnement et de faciliter la résolution du problème, au prix d'un temps de résolution plus élevé. Les méthodes basées sur le suivi de chemin ($PATH$ et $PATH+CS$) sont très peu affectées par l'écart type et la vitesse de l'utilisateur, dès lors que celui-ci parvient à la configuration finale.

3.3 Conclusion et Perspectives

Dans ce chapitre, nous avons décrit une méthode permettant de faire interagir efficacement un opérateur humain et un algorithme de planification de mouvement de type *RRT* pour déplacer

un objet dans le cadre d'une recherche de chemin. Cette méthode repose sur l'échange de pseudo-forces et d'indices visuels entre les deux parties via la scène virtuelle et les informations récoltées sur celle-ci après chaque détection de collision. Ces pseudo-forces sont d'une part calculées par l'algorithme pour aider l'utilisateur dans sa recherche et d'autre part générées par l'utilisateur à l'aide d'un périphérique interactif (souris 6D ou bras haptique) pour guider le développement du graphe. Nous avons vu sur des exemples simples et sur un cas industriel que la méthode permet d'accélérer la recherche de chemin par l'opérateur et bénéficie des avantages de la recherche automatique, à savoir une exploration de l'espace des configurations plus globale et l'amélioration du chemin approximatif de l'opérateur. L'analyse a montré que les différents paramètres pouvant influencer le temps de résolution peuvent être adaptés en fonction de la situation et de l'utilisateur. Le *IRRT* doit donc se concevoir comme une méthode se comportant différemment en fonction du problème, de l'utilisateur et des contraintes liées à la tâche.

Perspectives

Il serait intéressant d'étudier un peu plus l'influence des différents paramètres décrits dans la section 3.2.4.4 sur les performances de la recherche de chemin (en termes de nombre d'itérations et de nombre de nœuds dans le graphe) et sur la forme du chemin final (a-t-il une forme plutôt lisse comme le chemin tracé par l'opérateur ou l'aspect plus chaotique d'une ligne brisée comme un chemin trouvé automatiquement). Dans des cas complexes où il existe plusieurs solutions, il serait intéressant de comparer les solutions trouvées lorsque l'influence de l'opérateur est plus grande avec les solutions dans le cas où l'influence de l'algorithme est plus grande. Cela permettrait d'observer le respect des critères métier (le chemin trouvé est-il naturel, intuitive ou même faisable pour un humain ?) sur la recherche de chemin. Les environnements de type industriels sont des exemples adaptés à ce type de tests.

Le retour haptique exerce une influence sur les mouvements de l'opérateur humain (vitesse, précision). Une analyse des répercussions de cette influence sur les performances de l'algorithme permettrait de déterminer des paramètres acceptables pour le calcul du retour (notamment l'intensité de la force et le fait que les changements soient lisses et fluides). L'étude de tous ces paramètres serait aussi utile pour améliorer l'algorithme en les modifiant dynamiquement (lors de la recherche) en fonction de la topologie de l'environnement, de l'expérience de l'utilisateur et de l'avancement de l'exploration. En pratique, l'utilisateur n'a pas une vitesse constante et peut se trouver dans des cas où il aura moins de facilités que d'autres, dans des environnements variés où certains passages seront plus difficiles. Il serait donc intéressant de pouvoir changer de méthode d'échantillonnage et d'écart type au cours de la planification en fonction des difficultés qu'a l'utilisateur à résoudre un problème.

Il serait aussi profitable d'améliorer les aides fournies par l'algorithme à l'opérateur, tant visuelles que haptiques. Actuellement l'aide visuelle se fait uniquement par la coloration des

nœuds du graphe, coloration qui n'est par conséquent que ponctuelle. Une coloration de l'arrière plan ou par modification de la teinte (de façon continue en terme de pixels) de certaines zones peut être envisagée afin de rendre l'aide visuelle plus efficace et plus intuitive pour l'opérateur. Dans le même but, l'affichage en transparence d'une flèche ou d'autres métaphores visuelles fournirait une ou plusieurs directions préférentielles à suivre. D'autres types d'indices peuvent être envisagés, tels que ceux présentés dans [Sreng et al. 2006]. Couplé à ces affichages, un champ de forces calculé à partir des étiquettes des nœuds du graphe donnerait la même direction préférentielle. Cela permettrait de remplacer la pseudo-force unique calculée actuellement et qui permet juste de ramener l'utilisateur vers le graphe.

L'ajout des forces de contacts au retour haptique est aussi une possibilité envisagée, à condition qu'il soit accompagné de la modélisation des glissements et qu'il soit permis à l'opérateur d'activer ou de désactiver ces contacts manuellement durant la planification.

Concernant l'algorithme lui-même, d'autres méthodes d'échantillonnage peuvent être développées, qui combineraient les avantages des méthodes présentées dans ce chapitre. Par exemple, une méthode permettant d'échantillonner uniquement autour des parties du chemin utilisateur qui sont dans des zones non explorées par le graphe (zones de décrochage, ou en cas de retard de l'algorithme sur l'opérateur).

De manière plus générale, il serait intéressant d'augmenter le nombre de périphériques interactifs utilisés en même temps lors d'une recherche. Cela permettrait sur des problèmes très complexes de contrôler plus de degrés de liberté ou bien de cumuler l'aide de plusieurs opérateurs sur une même recherche. De même, augmenter le nombre de degrés de liberté pouvant être influencés par un seul périphérique interactif autoriserait la résolution de problèmes avec des systèmes plus complexes, tels que des personnages virtuels. Ce qui nous amène au sujet du prochain chapitre, dans lequel il sera aussi question d'une autre manière d'utiliser le planificateur interactif : la planification de mouvement en temps réel.

4

Le planificateur de locomotion interactif

4.1 Introduction

De nombreuses applications en planification de mouvement sont aujourd'hui explorées hors de la communauté robotique. L'une d'elle est l'animation graphique et particulièrement l'animation de personnages virtuels. Animer un personnage est une tâche complexe qui demande beaucoup de temps et d'expérience pour obtenir un mouvement qui semble naturel aux yeux de spectateurs. Pour cette raison, de nombreux travaux ont été effectués pour faciliter la tâche des animateurs de personnage. L'un des mouvements les plus difficiles à modéliser et des plus récurrents est la locomotion humaine, car la grande majorité des actions humaines nécessitent un déplacement. Pour sembler naturelle, la modélisation de la locomotion doit respecter plusieurs contraintes telles que l'équilibre et le contact avec le sol.

Le corps humain est classiquement modélisé dans le contexte de l'animation graphique par un ensemble de corps rigides articulés, comme par exemple sur la figure 4.1. Cette modélisation est aussi adoptée pour les robots humanoïdes. La résolution d'un problème de planification avec un tel modèle à l'aide d'algorithmes classiques (déterministes ou probabilistes) demande trop de temps pour être abordée telle quelle. Même si de nombreux travaux explorent la planification de mouvement corps complet pour un robot humanoïde ou un personnage virtuel [Yoshida et al. 2008; Kanehiro et al. 2008], les approches à deux niveaux découplant animation et navigation restent encore utiles car elles permettent de simplifier le problème en explorant un espace des

FIG. 4.1 – Eugène, le squelette virtuel utilisé dans les tests de ce chapitre. En bleu, les différentes articulations d'Eugène sont affichées, représentant 63 degrés de liberté au total.

configurations (*CS*) de dimension 3, laissant le soin de l'animation et le déplacement individuel de chaque articulation à d'autres techniques plus efficaces.

Bien que la locomotion humaine ne se limite pas à la marche (course, déplacements accroupi, à quatre pattes, etc.), celle-ci sera le mode de locomotion considéré dans ce chapitre, car elle est la plus utilisée dans les applications utilisant des personnages virtuels. De plus nous considèrerons des environnements structurés, artificiels et en intérieur tels que des bureaux, des maisons ou d'autres bâtiments, car ce sont des environnements récurrents, avec des formes simples à modéliser en général. Ils peuvent être représentés comme des successions de pièces larges et peu encombrées séparées par des ouvertures relativement étroites.

L'animation de marche pour un personnage virtuel se retrouve dans beaucoup d'applications telles que les films d'animation ou les jeux vidéos. Pour ces derniers une autre problématique se pose : l'animation en temps réel. En effet, dans des applications de type films d'animation ou simulations virtuelles d'humains, la planification et l'animation des mouvements peuvent être faites hors-ligne et sans fortes contraintes de temps. Dans des applications interactives, de type jeux vidéos, il est nécessaire de réagir dans un temps raisonnable aux actions d'un opérateur humain afin de suivre ses directives et en même temps d'animer le personnage de la manière la plus naturelle possible.

Dans un première partie nous décrirons différents travaux déjà effectués sur la même thématique ainsi que les travaux sur lesquels notre contribution est basée. Après avoir décrit

le principe de notre méthode et son implémentation, nous en montrerons les avantages à travers quelques exemples.

4.1.1 Travaux précédents

Beaucoup de travaux et d'études ont été menés sur l'animation de personnages virtuels que ce soit uniquement pour la locomotion ou l'animation de la totalité du personnage. Les premières méthodes de synthèse de mouvements humains ont d'abord été basées sur des connaissances biomécaniques [Zeltzer 1982]. La notion de positions clé [Burtnyk and Wein 1971; Sturman 1984] a ensuite été beaucoup utilisée en animation pour faciliter le travail des animateurs. Au lieu de créer une pose pour chaque image de l'animation, il s'agit de créer des positions intermédiaires entre lesquelles la trajectoire de chaque articulation est calculée automatiquement.

Avec l'arrivée de la capture de mouvement, beaucoup de nouvelles techniques d'animation se sont développées permettant de combiner réalisme et faible temps de calcul. La capture de mouvement permet de suivre et d'enregistrer les mouvements d'un sujet dans le but de les reproduire sur une cible virtuelle. Ces techniques se basent sur des méthodes de traitement du signal introduites dans [Bruderlin and Williams 1995]. Dans [Kovar et al. 2008] l'auteur introduit les graphes de mouvement. Ces graphes utilisent une base de captures de mouvement de différents types de marche pour construire un graphe reliant chaque mouvement (représenté par une arête) à un autre. Ces liaisons sont faites par l'intermédiaire de mouvements de transition et de nœuds de choix permettant de définir quels mouvements peuvent se suivre. Ce graphe est ensuite exploré pour générer des mouvements en respectant des contraintes définies par l'utilisateur.

Ensuite, plusieurs méthodes ont été développées permettant à la fois de prendre en compte les différentes possibilités du corps humain lors de l'animation et de naviguer à travers un environnement encombré en évitant les obstacles à l'aide d'algorithmes de planification de mouvement. Dans [Bottesi et al. 2004], les auteurs montrent l'intérêt d'utiliser des méthodes de planification de mouvement pour l'animation graphique en l'illustrant par un scénario de jeu vidéo. Dans [Salomon et al. 2003], l'auteur décrit un planificateur de mouvement permettant de se déplacer dans un environnement encombré et avec plusieurs niveaux d'élévation du sol. Le planificateur permet de prendre en compte le changement d'élévation, de contourner les obstacles à la manière d'un « Bug Algorithm » [Lumelsky and Stepanov 1987] et de passer par dessus des petits obstacles. Ce planificateur ne se charge pas d'animer la marche du personnage, l'animation pouvant être ajoutée à partir de la trajectoire trouvée après la planification.

Dans [Shiller et al. 2001], les auteurs planifient les mouvements d'un personnage virtuel à 34 *ddl* à partir d'une grille construite par décomposition cellulaire uniforme de l'environnement. Cette grille est construite pour différentes postures du personnage qui définissent les différents

types de locomotion envisagés, donnant ainsi une grille à plusieurs couches. Lors de la construction de cette grille, seules les boîtes englobantes des différentes postures sont prises en compte. Une trajectoire est ensuite trouvée à travers cette grille et les animations de locomotion pour chaque posture sont ajoutées à la trajectoire. Ces animations sont pré-enregistrées à l'aide de captures de mouvement. Pour finir, un filtre dynamique modifie l'animation pour obtenir un comportement plus réaliste.

Dans [Mombaur et al. 2010], les auteurs utilisent une méthode de commande optimale inverse afin de générer des trajectoires de locomotion humaines naturelles. Des trajectoires humaines sont prises grâce à un système de capture de mouvement. Elles sont considérées comme optimales et servent donc à définir le critère d'optimisation de la locomotion, critère utilisé pour générer les trajectoires.

Enfin, dans [Pettré 2003; Truong et al. 2010] les auteurs animent des personnages virtuels en combinant des méthodes de synthèse de mouvement avec des algorithmes probabilistes de planification de mouvement. C'est en se basant sur cette méthode, décrite dans la section suivante, que nous avons développé un planificateur interactif de marche pour personnages virtuels. D'autres travaux combinant planification de mouvement et locomotion de personnages virtuels sont présentés dans [Choi et al. 2003; Lau and Kuffner 2005]. Des travaux permettant de synthétiser des mouvements à partir de bibliothèques de captures de mouvement sont présentées dans [Arikan and Forsyth 2002].

L'objectif de ce chapitre n'est pas d'étudier des méthodes de contrôle de la locomotion d'un personnage virtuel, mais plutôt d'appliquer la méthode de planification interactive développée dans le chapitre précédent à ces méthodes. Des références plus complètes concernant la locomotion d'un personnage virtuel peuvent être trouvées dans des travaux dédiés à ce domaine tels que [Boulic 2005; Sud et al. 2007; Sud et al. 2007; Glardon 2006; Fuchs and Donikian 2009].

4.1.2 Le contrôleur de locomotion

Pour animer notre personnage virtuel après l'obtention d'un chemin grâce à un algorithme de planification de mouvement, nous nous basons sur les travaux qui vont être décrits dans cette section.

4.1.2.1 Planification de mouvement de marche pour Acteurs Digitaux

Dans [Pettré 2003], l'auteur introduit une méthode de planification de mouvement de marche pour un personnage virtuel. Cette méthode comporte une étape de planification de trajectoire qui définit la trajectoire suivie par le personnage dans la scène virtuelle et une étape d'animation qui permet de donner au déplacement un aspect de marche naturelle. Une dernière étape qui consiste à synthétiser des mouvements d'évitement d'obstacles par la partie supérieure

du corps est aussi décrite dans ces travaux, mais nous ne la retranscrirons pas ici car elle ne sera pas utilisée.

La planification de trajectoire

Dans la suite de ce chapitre, on appellera **trajectoire** un chemin sur lequel sont ajoutées des informations de vitesse. L'étape de planification de trajectoire permet d'obtenir la trajectoire que le personnage virtuel empruntera pour satisfaire la requête de l'utilisateur, requête exprimée par une configuration initiale et une configuration finale. Le personnage virtuel pouvant posséder un nombre important de *ddl*, une méthode classique de planification appliquée directement à ce problème n'est pas envisageable. Le problème a donc été réduit au déplacement d'une boîte englobant le personnage à la surface de l'environnement considéré. Nous nous ramenons alors à un problème de planification pour un *CS* de dimension 3 paramétré par x, y et θ.

Afin de pouvoir satisfaire plusieurs requêtes de l'utilisateur sur le même environnement, une méthode de type *PRM* a été choisie. Il s'agit de la méthode « *PRM* de visibilité » (section 2.2.2). L'intérêt de cette méthode dans le contexte de l'animation est qu'elle permet d'effectuer des recherches de chemin très rapidement, afin de libérer du temps de calcul pour l'animation. Mais elle nécessite la construction d'un graphe au préalable, étape coûteuse qui est faite hors-ligne et qui permet d'avoir un graphe représentant une bonne couverture de CS_{free}.

La méthode locale, qui permet de relier les nœuds du graphe entre eux par un chemin sans collisions, a été choisie pour permettre à l'algorithme de planification de fournir des profils de déplacement d'aspect naturel adapté à la problèmatique de la locomotion. Les chemins locaux dans le graphe sont ainsi modélisés par des courbes de Bézier de degré 3 (figure 4.2) qui permettent d'obtenir un chemin lisse et relativement naturel dès la résolution de la planification. Ces courbes sont décrites par une équation paramétrée (équation 4.1) dont les paramètres sont les coordonnées des 4 points de contrôle.

$$B(u) = \sum_{k=0}^{3} P_k \frac{3!}{k!(3-k)!} u^k (1-u)^k \tag{4.1}$$

Les principaux avantages des courbes de Bézier, qui en font une méthode locale valide sont les suivants :

- Les courbes passent par leurs points de contrôle extrêmes. Dans le cadre d'une méthode locale, les points de contrôle extrêmes sont les deux configurations que l'on cherche à relier. Il est donc nécessaire que le chemin local passe par ces deux configurations.
- Le chemin créé reste dans l'enveloppe convexe des points de contrôle. En choisissant bien les points de contrôle manquants pour une courbe de Bézier de degré 3, on peut minimiser l'encombrement de ce chemin.
- La courbe est tangente à la direction définie par les deux premiers points de contrôle, ainsi qu'à celle définie par les deux derniers.

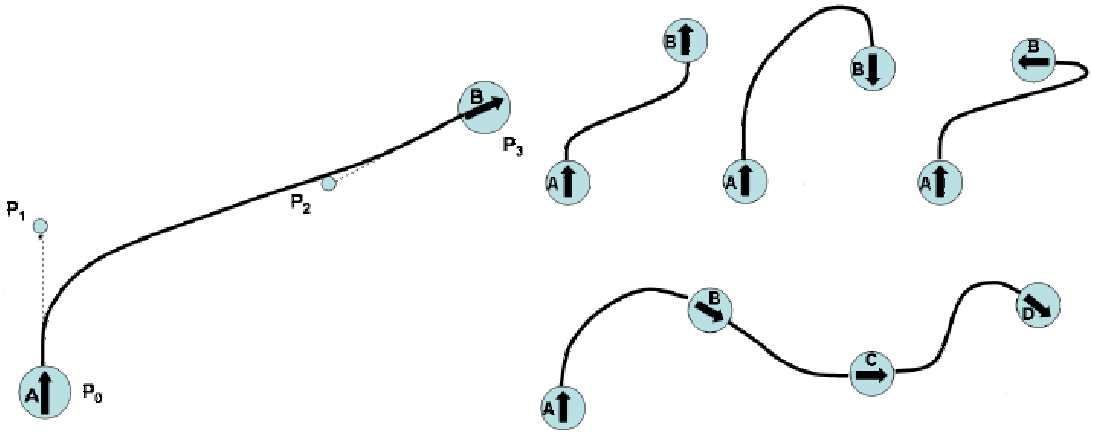


FIG. 4.2 – De [Pettré 2003]. Quelques exemples de chemins locaux construits avec des courbes de Bézier. En bas à droite, un exemple de chemin global composé de plusieurs chemins locaux de Bézier.

– La courbe et sa dérivée sont continues.

Cette méthode locale sera utilisée dans le planificateur interactif décrit dans ce chapitre. L'étape de planification de trajectoire peut être décomposée en deux parties :

– Une partie de planification de chemin, sans information de vitesse, où l'on obtient un chemin libre de collision entre les configurations initiale et finale. Avec le graphe obtenu, il est possible de répondre aux requêtes de l'utilisateur (c'est-à-dire trouver un chemin entre les configurations initiale et finale données). Le chemin obtenu est constitué d'une succession de courbes de Bézier, il est lisse et faisable par le personnage virtuel, mais pas nécessairement optimal, comme on peut s'y attendre avec des algorithmes probabilistes. Pour obtenir un chemin plus intuitif pour un être humain il est nécessaire de rajouter une phase d'optimisation qui permet d'améliorer l'aspect global du chemin (pas de déplacements de type "zig-zag").
– Une partie de transformation du chemin obtenu en trajectoire. Afin de pouvoir animer le personnage virtuel le long de ce chemin, l'auteur définit un profil de vitesse le long de celui-ci, qui permettra de choisir les captures de mouvement sources dans la bibliothèque de captures de mouvement enregistrées. L'exécution du chemin a été voulue comme la plus rapide possible, tout en respectant les limites de vitesse (linéaire et angulaire) et d'accélération tangentielle imposées par le contenu de la bibliothèque.

L'animation de la trajectoire

L'animation de la trajectoire obtenue est basée sur une technique d'interpolation de captures de mouvements. L'auteur décrit donc une méthode pour enregistrer et stocker de façon exploitable les animations capturées. Chaque animation capturée est effectuée à vitesse constante, que ce soit pour la vitesse linéaire ou la vitesse angulaire. Un traitement automatique

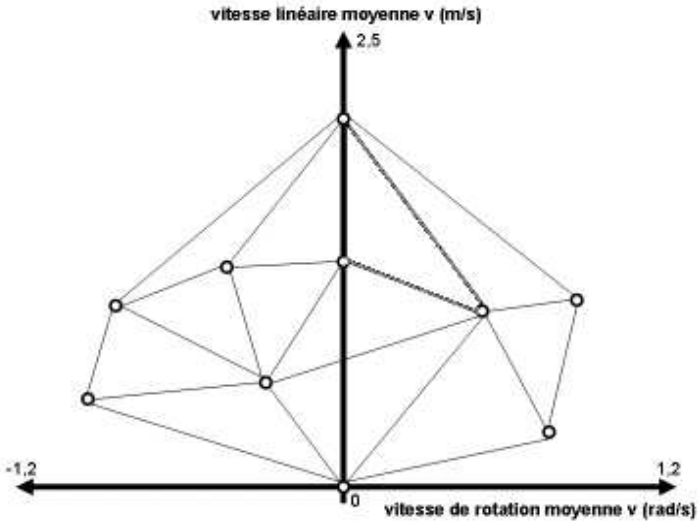


FIG. 4.3 – De [Pettré 2003]. Représentation de l'espace (v,ω). Chaque point blanc représente une capture de mouvement, caractérisée par sa vitesse linéaire et sa vitesse angulaire. L'ensemble de mouvements qui peuvent être générés à partir de ces captures se situe à l'intérieur du polygone convexe formé par ces points.

des captures de mouvement est effectué pour s'assurer que toutes les animations présentent les caractéristiques nécessaires à leur exploitation. Ainsi, chaque capture doit être caractérisée par les vitesses moyennes du pelvis que ce soit pour la vitesse linéaire ou la vitesse angulaire. Une capture de mouvement est représentée dans l'espace (v,ω) avec v la vitesse linéaire et ω la vitesse angulaire (figure 4.3).

Ces vitesses caractéristiques sont le lien entre le chemin planifié et l'animation du personnage. Il s'agit de projeter dans l'espace (v,ω) tous les échantillons de la trajectoire calculée lors de la phase de planification, grâce à ses vitesses linéaire et angulaire. Pour une vitesse désirée d'un échantillon de la trajectoire, on détermine la séquence de mouvement en projetant cette vitesse dans (v,ω) et en interpolant parmi les captures de la bibliothèque de mouvement. Ce processus est effectué à l'aide d'une triangulation de Delaunay [Delaunay 1934]. Trois animations candidates sont extraites, à partir desquelles on interpole les valeurs de chaque *ddl* de l'échantillon concerné. Une dernière étape permet d'adapter l'animation aux extrémités afin que les configurations de début et de fin fournies par l'utilisateur soient respectées.

La combinaison d'une phase de planification de mouvement et d'une phase d'animation du personnage virtuel permet de générer des mouvements de marche avant réalistes dans des environnements encombrés tout en évitant les obstacles présents.

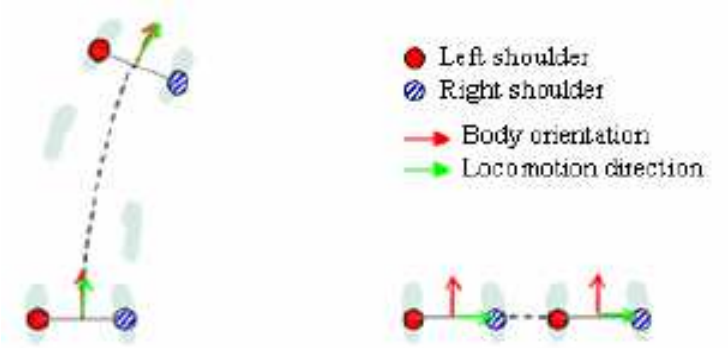


FIG. 4.4 – De [Truong et al. 2010]. A gauche un exemple de déplacement non-holonome pour un humain. A droite, un déplacement holonome.

4.1.2.2 Locomotion holonome et non-holonome

Dans la méthode décrite dans la section précédente, les auteurs ne prennent en compte qu'un des aspects de la locomotion humaine puisqu'elle est exclusivement limitée à la marche avant. Dans [Truong et al. 2010], les auteurs observent que les trajectoires de la marche humaine peuvent être distinguées en deux classes principales : les trajectoires *holonomes* et les trajectoires *non-holonomes*. La notion de contrainte non-holonome pour une trajectoire peut être illustrée par le déplacement d'une voiture. En effet, une voiture ne peut pas se déplacer latéralement. Pour atteindre une configuration située sur le coté, avec la même orientation, il lui faut faire plusieurs manœuvres (exemple : le créneau pour se garer). L'humain peut effectuer de tels déplacements par un pas de côté, ce qui indique un comportement holonome, en plus du comportement non-holonome pour la marche avant (figure 4.4). Les différentes trajectoires humaines peuvent être classées dans ces deux groupes en observant l'orientation du corps par rapport à la direction de déplacement. Les auteurs ont donc développé une méthode se basant sur le principe du contrôleur de locomotion de [Pettré 2003] permettant de prendre en compte des déplacements holonomes.

Pour prendre en compte un déplacement holonome, il faut ajouter une troisième vitesse pour caractériser les animations qui va engendrer des vitesses linéaires latérales, nommées v_l. Cette nouvelle caractérisation permet de synthétiser un plus large champ de mouvements en intégrant la marche de côté. Chaque configuration de la trajectoire calculée pour la racine du personnage est ainsi projetée dans l'espace (v, ω, v_l) de dimension 3 et les valeurs angulaires du personnage sont interpolées à partir de 4 captures de mouvements différentes (voir figure 4.5).

Dans tous les travaux que nous avons décrit dans cette section, les algorithmes de planification de mouvement n'étaient utilisés que dans une phase préalable à l'animation. L'utilisateur effectuait une requête de planification et une fois la trajectoire solution trouvée, elle était animée à l'aide d'un contrôleur de locomotion et d'une base de captures de mouvements.

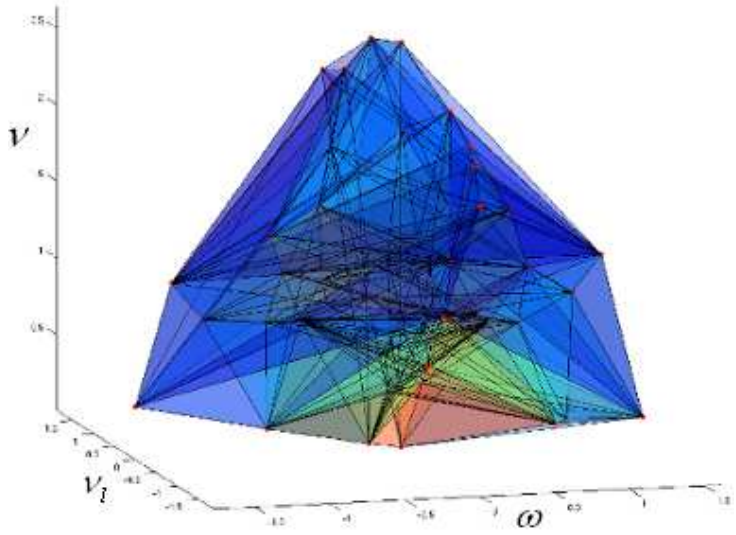


FIG. 4.5 – De [Truong et al. 2010]. Représentation de l'espace (v,ω, v_l). Tout les échantillons dans l'espace sont structurés en nuage de tétraèdre grâce à une triangulation 3D. Chaque sommet d'un tétraèdre représente une capture de mouvement caractérisée par sa vitesse linéaire, sa vitesse angulaire et sa vitesse latérale.

L'utilisation d'un méthode de type *PRM* impliquait une exploration à priori de l'environnement. Dans la section suivante, nous nous sommes basés sur ces travaux pour développer une méthode permettant d'effectuer une animation de personnage de manière interactive.

4.2 Le planificateur de locomotion Interactif

L'idée principale du planificateur de locomotion interactif, nommé *ILP* (pour *Interactive Locomotion Planner*) est de combiner les avantages du planificateur interactif décrit dans le chapitre 3 et nommé *IRRT*, avec les avantages du contrôleur de locomotion de [Pettré 2003] et [Truong et al. 2010] : réduire un problème de planification de marche humaine compliqué (63 *ddl* pour Eugène) à un problème de planification de mouvement simple (3 *ddl*) guidé par un utilisateur humain tout le long de l'exécution.

Dans une première étape, nous avons effectué une version découplée du planificateur. Les étapes de planification et d'animation restent bien distinctes. La planification se fait grâce à la méthode *IRRT* et la trajectoire trouvée est ensuite animée. Cela permet de résoudre des requêtes de planification classiques, avec en entrée une configuration initiale et une configuration finale. Dans une seconde étape nous avons choisi de développer une méthode combinant planification et animation. La particularité de cette méthode est qu'elle ne nécessite pas de configuration finale en entrée. La planification et l'animation sont effectuées au fur et à mesure du déplacement de l'avatar par l'utilisateur.

4.2.1 Le planificateur semi-interactif

L'objectif du planificateur de locomotion semi-interactif est de pouvoir effectuer une requête de planification ayant un but particulier. Dans ce contexte, l'utilisateur doit donner en entrée le modèle de l'environnement, celui d'Eugène, une configuration initiale et une configuration finale. La bibliothèque de capture de mouvement est fournie a priori et il s'agit de celle décrite dans [Truong et al. 2010], avec 68 enregistrements de différents mouvements de locomotion, composés de marches avant, de pas de côté et de virages combinés de différentes façons et à différentes vitesses.

Algorithm 5 Algorithme semi-interactif

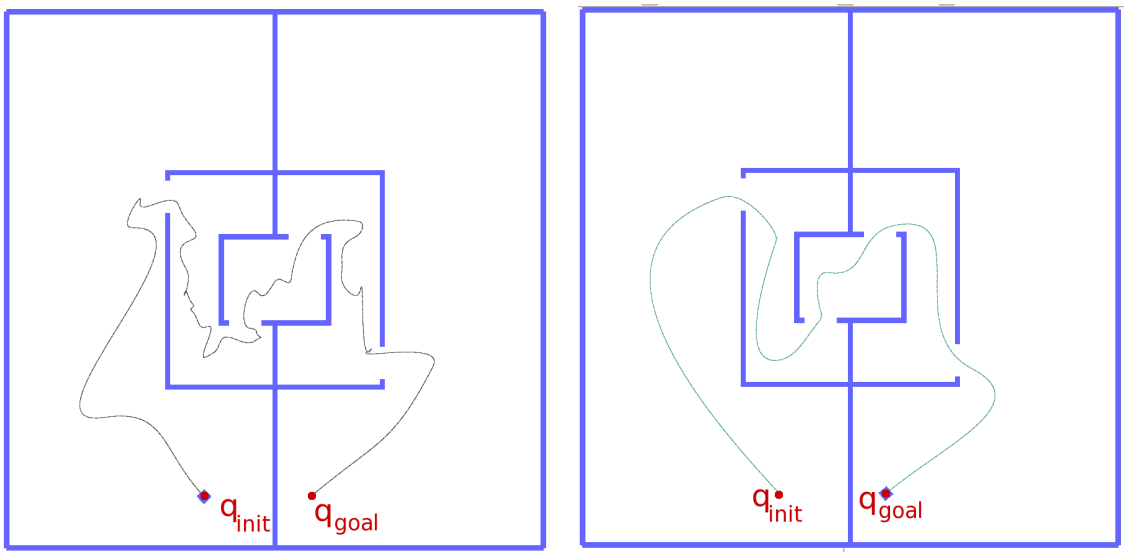
$P_{bezopt} \leftarrow$ IRRT(q_{init},q_{goal})
$T_{root} \leftarrow$ PATH_TO_TRAJ(P_{bezopt})
$T_{anim} \leftarrow$ ANIMATE(T_{root})

Le planificateur, décrit dans l'algorithme 5, peut se décomposer en trois étapes principales :

- Une étape de planification de mouvement avec la méthode *IRRT*. Cette étape se déroule telle que décrite dans le chapitre 3, avec une modification au niveau de la méthode locale et du calcul de distance, pour rendre le chemin final plus naturel. Cette étape est effectuée avec une boîte englobante possédant 2 ddl en translation et 1 ddl en rotation (X, Y, θ).
- Une étape de transformation du chemin P_{bezopt} en trajectoire. Pour cela le chemin est ré-échantillonné et des informations de vitesse et d'accélération sont ajoutées à chaque nouvel échantillon (PATH_TO_TRAJ). La trajectoire T_{root} obtenue dans un CS de dimension 3, est celle du pelvis du personnage virtuel (située au niveau du pelvis pour Eugène). On utilise pour cela le même principe que dans la thèse de [Pettré 2003].
- Une étape d'animation de T_{root}. Pour chaque échantillon de la trajectoire, la posture de Eugène lorsqu'il est dans cette configuration est calculée à partir des captures de la bibliothèque fournie en entrée. La trajectoire T_{anim} obtenue contient les valeurs de tous les ddl du personnage virtuel (CS est de dimension 63 pour Eugène).

Un chemin plus « naturel »

Dans le chapitre 3, nous avons décrit la méthode *IRRT* en supposant un système de type « *free-flyer* » dans lequel l'avatar peut se déplacer de manière holonome dans CS. Même si un être humain peut aussi se déplacer de manière holonome, la plupart du temps il utilise une marche non-holonome pour ses déplacements. Dans un premier temps, nous avons donc utilisé les courbes de Bézier pour modéliser un comportement « naturel » de marche entre deux configurations. La méthode locale développée pour créer ces courbes est la même que dans [Pettré 2003]. L'orientation θ du personnage virtuel doit être tangente à la courbe de Bézier. Le calcul de l'orientation le long du chemin local est donc effectué dans la méthode locale.



(a) (b)

FIG. 4.6 – Une planification non-interactive effectuée avec (a) une méthode locale de Bézier normale et (b) une méthode locale de Bézier modifiée.

L'orientation des configurations extrémales du chemin local demeure libre (*CS* est de dimension 3). Cette méthode locale ne permet pas de faire apparaître l'aspect holonome de la locomotion. Pour faire apparaître cet aspect dans le planificateur semi-interactif, nous utilisons la méthode de transformation d'un chemin linéaire en courbe de Bezier décrite dans le planificateur interactif (section 4.2.2 et suivante).

Néanmoins, dans le cadre d'algorithmes de diffusion, cette méthode permet d'avoir des chemins qui semblent naturels uniquement localement (figure 4.6-(a)). Afin de rendre ce critère plus global et d'avoir une trajectoire finale à l'aspect plus naturel, nous avons modifié la façon dont les chemins locaux sont construits et plus particulièrement le choix des configurations qui sont liées entre elles par un chemin local. Cette modification est faite par l'intermédiaire du choix du plus proche voisin à connecter au nouvel échantillon. Le calcul de la distance a donc été modifié pour favoriser la marche avant. Trois étapes sont nécessaires pour cela :

- Les rotations étant prises en compte dans le calcul des distances dans *CS*, nous avons appliqué un coefficient w à celles-ci pour augmenter leur influence. Ainsi, pour une différence de positions égale dans l'espace des configurations, ce sera la configuration dont l'orientation est la plus proche de l'échantillon qui sera choisie comme voisin le plus proche. La nouvelle distance prenant en compte w est appelée d_w (figure 4.8-(b)).
- Un coefficient o est ensuite appliqué à d_w. Ce coefficient prend en compte la position relative des deux configurations q_1 et q_2 que l'on tente de lier ensemble. Plus l'orientation de q_1 est éloignée de la direction q_1q_2, plus la distance est grande. La nouvelle distance

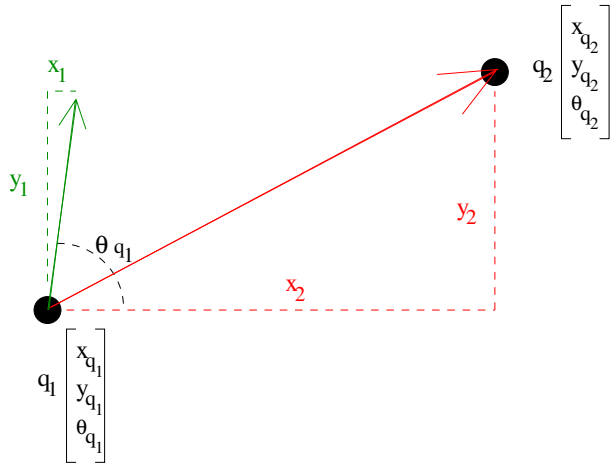


FIG. 4.7 – En rouge le vecteur $q_1 - q_2$ représenté par x_2 et y_2. En vert le vecteur unitaire correspondant à l'orientation de q_1, représenté par x_1 et y_1. En noir, les configurations q_1 et q_2 représentées par leurs coordonnées (x_{q_i}, y_{q_i} et θ_{q_i})

obtenue d_o est la distance utilisée pour le calcul du plus proche voisin (figure 4.8-(c)). Le calcul de d_o est détaillé dans l'équation 4.2 dans laquelle $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ est le vecteur unitaire qui représente l'orientation θ_{q_1} de q_1, $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ est le vecteur $q_1 - q_2$ et (x_{q_i}, y_{q_i},θ_{q_i}) sont les coordonnées dans *CS* de la configuration q_i (voir figure 4.7 pour plus de précisions).

– Une étape d'optimisation, en fin de planification, permet d'obtenir un chemin plus réaliste. Les étapes précédentes permettent d'obtenir un chemin d'aspect plus naturel qu'un simple chemin en ligne brisée, mais il a toujours un aspect « sinueux ». Cette étape d'optimisation permet d'obtenir un chemin plus « direct » entre la configuration initiale et la configuration finale.

$$d_w = \sqrt{(x_{q_2} - x_{q_1})^2 + (y_{q_2} - y_{q_1})^2 + w \times (\theta_{q_2} - \theta_{q_1})^2} \quad (4.2)$$

$$o = e^{-(x_1 \times x_2 + y_1 \times y_2)} \quad (4.3)$$

$$d_o = d_w \times o \quad (4.4)$$

Avec une méthode locale basée sur les courbes de Bézier et ce calcul de distance particulier, le planificateur *ILP* semi-interactif permet donc de créer des trajectoires qui sont naturelles pour un être humain, non seulement au niveau local, mais aussi au niveau de la trajectoire globale elle-même (figure 4.6-(b)). Dès la phase de planification terminée, l'animation de Eugène le long de la trajectoire obtenue se fait comme indiqué dans l'algorithme 5. Cette méthode est la première façon d'utiliser le planificateur de locomotion interactif, nous allons dans une seconde

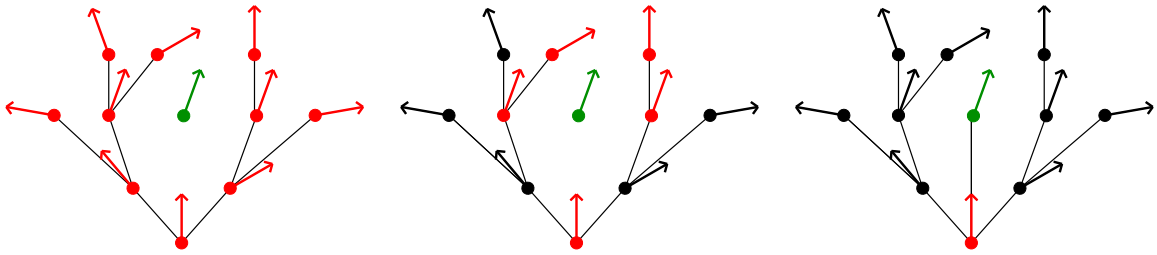


FIG. 4.8 – En rouge, les nœuds de l'arbre et leur orientation pouvant potentiellement être choisis comme plus proche voisins du nouvel échantillon (en vert). (a) En cas de calcul de distance classique, tous les nœuds de l'arbre peuvent être choisis. (b) En ajoutant un poids plus important sur les rotations, les nœuds rouges sont plus proches que les nœuds noirs. (c) En utilisant la distance d_o, c'est le nœud rouge qui est choisi comme plus proche voisin.

partie présenter une version interactive du planificateur.

4.2.2 Le planificateur interactif

Le planificateur de locomotion interactif a pour objectif de permettre le « contrôle augmenté » d'un personnage virtuel. L'utilisateur contrôle la direction de déplacement du personnage et la partie algorithmique permet l'évitement automatique des obstacles, tout en générant l'animation de marche du personnage le long de la trajectoire tracée par l'utilisateur.

Le planificateur interactif n'a donc pas besoin de configuration finale en entrée (mais il est possible de lui en donner une). Les seules données nécessaires avant la planification sont une configuration initiale et un modèle de l'environnement et du personnage virtuel. Comme pour le *ILP* semi-interactif, la bibliothèque de capture est fournie à priori.

Algorithm 6 Algorithme interactif

loop
  $P_{linear} \leftarrow$ IRRT(q_{pv})
  $P_{opt} \leftarrow$ BEZIER_OPTIMIZER(P_{linear})
  $T_{root} \leftarrow$ PATH_TO_TRAJ(P_{opt})
  $T_{anim} \leftarrow$ ANIMATE(T_{root})
  $q_{pv} \leftarrow$ PLAY(T_{anim})
end loop

Le *ILP* interactif, décrit dans l'algorithme 6, peut se décomposer en cinq étapes :

– Une étape de planification de mouvement avec l'algorithme *IRRT*. Cette étape se déroule telle que décrite dans le chapitre 3 à la seule différence que la planification se fait entre la configuration courante du personnage virtuel et la configuration courante de l'avatar

déplacé par l'utilisateur. Ici, la méthode locale utilisée est une méthode linéaire pour satisfaire les contraintes de temps.
– Une étape de transformation du chemin trouvé. Cette transformation se fait à l'aide d'une méthode basée sur les courbes de Bézier. Cette étape est nécessaire pour rendre le chemin final plus réaliste.
– Une étape de transformation du chemin en trajectoire. Tout comme dans l'algorithme *ILP* semi-interactif le chemin est ré-échantillonné et des informations de vitesse et d'accélération sont ajoutées à chaque nouvel échantillon.
– Une étape d'animation de la trajectoire où on interpole les captures de mouvements à partir des vitesses de chaque échantillon.
– Une étape de lecture de l'animation. Cette étape est nécessaire pour redéfinir la configuration courante du personnage virtuel à chaque itération de l'algorithme *ILP* et pour avoir un affichage de son déplacement à l'écran.

Cette succession d'étapes est répétée jusqu'à ce que l'utilisateur arrête la planification explicitement. Après l'étape 5, le planificateur reprend à l'étape 1 avec les nouvelles configurations du personnage virtuel et de l'avatar. Les paragraphes suivants permettent de décrire ces différentes étapes.

Le déplacement de l'avatar

Afin que la planification et l'animation du personnage virtuel puissent être effectués à un rythme interactif (l'utilisateur perçoit clairement son contrôle sur le déplacement du personnage), toutes les étapes de l'algorithme *ILP* doivent être effectuées rapidement, idéalement en temps réel. Dans ce but, il est nécessaire de contrôler la vitesse de déplacement de l'avatar, c'est à dire la sensibilité du périphérique interactif. De manière générale, le déplacement de l'avatar par l'opérateur durant le temps d'une itération sera petit par rapport à la taille de l'environnement, car la durée d'une itération de l'algorithme *ILP* est voulue la plus petite possible (de l'ordre de 1 Hz). Mais pour pouvoir exploiter les avantages de la planification de mouvement (évitement d'obstacles et chemin réaliste), la longueur du déplacement ne doit pas être trop petite. Un bon moyen de contrôler cette longueur est la sensibilité du périphérique interactif notée s_{pi}. Celle-ci correspond au rapport entre l'intensité de la pseudo-force exercée par l'utilisateur sur le périphérique et le déplacement que cela génère sur l'avatar de la scène virtuelle.

La phase de planification

Dans l'algorithme *ILP* interactif, le personnage virtuel est différent de l'avatar piloté par l'utilisateur. En effet, les étapes de planification et d'animation introduisent un temps de latence entre le moment où l'utilisateur déplace l'avatar et le moment où le personnage virtuel est effectivement déplacé. La totalité des cinq étapes de l'algorithme 6 devant se faire le plus

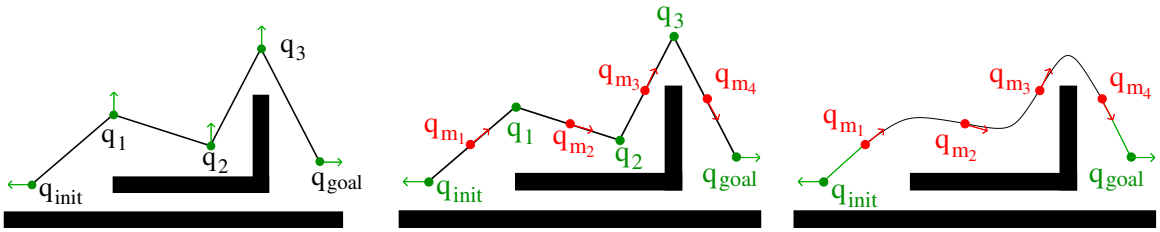


FIG. 4.9 – (a) Chemin obtenu après l'étape de planification de mouvement de l'algorithme *ILP* interactif. (b) A partir de ce chemin sont ajoutées les configurations situées au centre de chaque chemin local et on les oriente le long du chemin. Les orientations de q_{init} et q_{goal} sont conservées. (c) Les configurations q_i sont éliminées et des chemins locaux de Bézier sont créés entre les q_{m_i}. En vert apparaissent les parties holonomes de la trajectoire.

rapidement possible pour avoir un déplacement d'Eugène à l'écran dans un temps raisonnable, nous avons imposé deux simplifications à l'algorithme *IRRT* par rapport à la version semi-interactive de l'algorithme *ILP* :

- La méthode locale est linéaire. Le calcul des courbes de Bézier demande un temps important et le calcul particulier des plus proches voisins introduit dans l'algorithme *ILP* semi-interactif augmente le temps de résolution par rapport à un calcul classique. Pour accélérer la phase de planification, ces calculs sont supprimés. Les avantages que ces calculs procuraient sont repris lors de la phase de transformation.
- L'orientation θ de l'avatar n'est plus prise en compte. La modification de l'orientation le long de la trajectoire planifiée ici sera faite lors de la phase de transformation.

De plus, étant donné qu'il n'y a pas de configuration finale en entrée, la phase de planification (IRRT(q_{pv})) doit être arrêtée après un certain temps t_{planif} pour permettre de réaliser les autres phases. Ce temps est déterminé a priori, en fonction des performances de l'algorithme *ILP* voulues.

La phase de transformation

La phase de planification permet d'obtenir un chemin sans collisions entre la configuration courante du personnage virtuel, qui sera nommée q_{pv} et la configuration courante de l'avatar q_a. Ce chemin est formé de chemins locaux linéaires concaténés les uns aux autres, ce qui en fait une ligne brisée. Ce type de chemins n'est pas réaliste dans le cadre de l'animation d'un personnage. Une phase de transformation (BEZIER_OPTIMIZER(P_{linear})) a donc été développée[1] dans le but de rendre ce chemin plus réaliste.

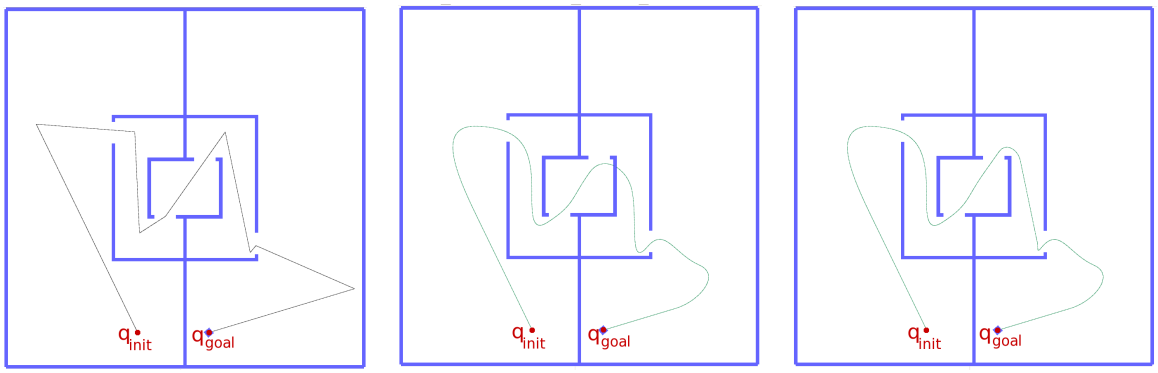
Cette étape est décrite dans l'algorithme 7 dans lequel K représente le nombre de noeuds contenus dans le chemin P_{linopt}. Elle peut être décomposée en 5 phases :

[1]Travail en collaboration avec *Antonio El Khoury*

Algorithm 7 BEZIER_OPTIMIZER(P_{linear})

$P_{linopt} \leftarrow$ STANDARD_OPTIMIZER(P_{linear})
$P_{opt} \leftarrow$ INIT_CONFIG(P_{linopt})
for $i = 1$ to $K-1$ **do**
  $q_i \leftarrow$ PATH_CONFIG(P_{linear} , i)
  $P_{opt} \leftarrow q_i$
end for
$P_{opt} \leftarrow$ GOAL_CONFIG(P_{linear})
for $i = 0$ to $K-2$ **do**
  $q_{m_i} \leftarrow$ MIDDLE_CONFIG(q_i , q_{i+1})
  $q_{m_{i+1}} \leftarrow$ MIDDLE_CONFIG(q_{i+1} , q_{i+2})
  REORIENT(q_{m_i})
  $P_{opt} \leftarrow q_{m_i}$, $q_{m_{i+1}}$
  $P_{opt} \leftarrow$ BEZIER_LOCAL_PATH(q_{m_i},$q_{m_{i+1}}$)
end for
$P_{opt} \leftarrow$ LINEAR_LOCAL_PATH(q_0,q_{m_0})
$P_{opt} \leftarrow$ LINEAR_LOCAL_PATH($q_{m_{N-1}}$,q_N)

– Une première phase d'optimisation (STANDARD_OPTIMIZER(P_{linear})). Cette optimisation est effectuée sur le chemin linéaire P_{linear} généré par l'algorithme *IRRT* afin d'avoir un chemin plus court entre les configurations initiale et finale. Sur la figure 4.9-(a), on peut voir les configurations intermédiaires, avec dans ce cas une orientation fixe.
– Une phase où les configurations q_{m_i} situées au milieu des segments reliant les configurations q_i et q_{i+1} sont crées en privilègiant des déplacements qui respectent la direction du chemin.
– Une phase où l'orientation θ de chaque configuration q_{m_i} du chemin optimisé est modifiée pour être colinéaire à la direction $q_i q_{i+1}$ (REORIENT(q_{m_i})). Cette phase permet ensuite de calculer correctement les points de contrôle de la courbe, afin que celle-ci suive la trajectoire désirée (figure 4.9-(b)).
– Une phase où de nouveaux chemins locaux sont créés entre les configurations q_{m_i} situées au centre des chemins locaux de P_{linopt}, à l'aide d'une méthode locale de Bézier (BEZIER_LOCAL_PATH(q_{m_i},$q_{m_{i+1}}$)). Sur la figure 4.9-(c) on peut voir les différents chemins locaux créés avec des courbes de Bézier entre q_{m_1} et q_{m_4}. Cette méthode intègre la vérification de collision le long du chemin local créé. Si ce chemin est en collision, alors q_{m_i} et $q_{m_{i+1}}$ sont déplacés vers q_i jusqu'à qu'il n'y ait plus de collisions (figure 4.10-(b) et (c)).
– Une dernière phase où les configurations initiale q_0 et finale q_N sont reliées au chemin final optimisé P_{opt} à l'aide d'une méthode locale linéaire (LINEAR_LOCAL_PATH). Pour cette dernière phase, les orientations d'origine de q_0 et q_N sont respectées, afin de satisfaire la requête originale de l'utilisateur. Le premier chemin local, ainsi que le



(a) (b) (c)

FIG. 4.10 – (a) Le chemin linéaire optimisé P_{linopt}. (b) Le chemin créé avec la méthode de transformation par courbes de Bézier, sans détection de collision. (c) Le chemin créé avec la méthode de transformation par courbes de Bézier, avec détection de collision.

dernier sont donc des déplacements holonomes, intégrés à une trajectoire globalement non-holonome. Cette implémentation suit une partie des observations effectuées dans [Truong 2010]. Sur la figure 4.9-(c) ces parties holonomes sont représentées en vert.

La transformation en trajectoire

Le chemin P_{opt} obtenu doit ensuite être transformé en trajectoire (PATH_TO_TRAJ(P_{opt})) avant de pouvoir être animé. Cette étape décomposée en 6 sous-étapes :

- La première sous-étape permet d'initialiser les limites de vitesse et d'accélération, nommées V_{max}, Ω_{max} et A_{max}, ainsi que la fréquence d'échantillonnage T.
- La seconde sous-étape permet de placer les échantillons le long de la trajectoire en respectant les limites de vitesse linéaire maximale et d'accélération maximale.
- La troisième sous-étape permet de placer les échantillons le long de la trajectoire en respectant la limite de vitesse angulaire maximale.
- La quatrième sous-étape consiste à vérifier si la contrainte d'accélération maximale est toujours respectée après les modifications faites à la troisième sous-étape.
- La cinquième sous-étape permet, en cas de violation de la contrainte à l'étape précédente, de calculer la distance de décélération nécessaire pour respecter la contrainte.
- La dernière sous-étape permet de calculer la décélération à appliquer, d'en déduire les vitesses de chaque échantillon et donc le placement des échantillons le long de la trajectoire.

L'animation

L'étape d'animation (ANIMATE(T_{root})) se fait en interpolant les captures de la bibliothèque de mouvement. Les captures de mouvement à interpoler sont choisies en fonction des vitesses

calculées à l'étape précédente et des vitesses propres de chaque capture de l'animation. Les quatre captures utilisées sont celles qui forment le tétraèdre qui contient le point représentant la vitesse de l'échantillon que l'on veut animer, dans l'espace (v, ω, v_l,).

La lecture de l'animation

Par défaut, le modèle géométrique du personnage virtuel n'est pas déplacé dans la scène. Mais le fait que l'utilisateur aie un retour visuel de ce déplacement permet un certain confort et devient nécessaire dans le cadre du planificateur *ILP* interactif, car l'algorithme *IRRT* utilise la configuration courante de ce modèle en tant que configuration initiale pour la partie planification de mouvement. À chaque itération, Eugène est donc déplacé d'un nombre de configurations n_{cfg} (PLAY(T_{anim})) qui dépend de la vitesse de chaque configuration sur la trajectoire à animer.

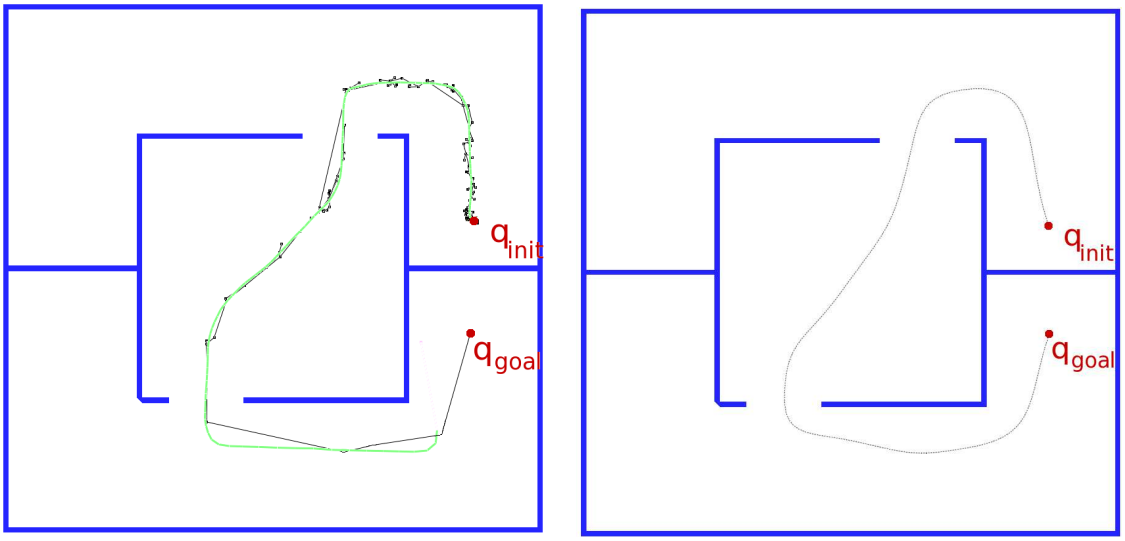
4.2.3 Quelques Exemples

Cet algorithme a été implémenté en C++ sur la base de la plateforme HPP (Humanoïd Path Planner) développée au LAAS et basée elle-même sur KineoWorks. Les expérimentations ont été faites sur un PC Dual Core, 2,1 GHz avec 2Go de RAM. Pour intégrer les parties holonomes du déplacement du personnage virtuel, nous avons utilisé pour la planification une interpolation linéaire en tant que méthode locale et appliqué le procédé de transformation du chemin par courbes de Bézier décrit à la section 4.2.2 pour le planificateur semi-interactif. La phase de planification se fait à partir de la boîte englobante du personnage virtuel pour les deux versions du planificateur et le périphérique d'interaction utilisé est une souris 6D.

4.2.3.1 Le planificateur semi-interactif

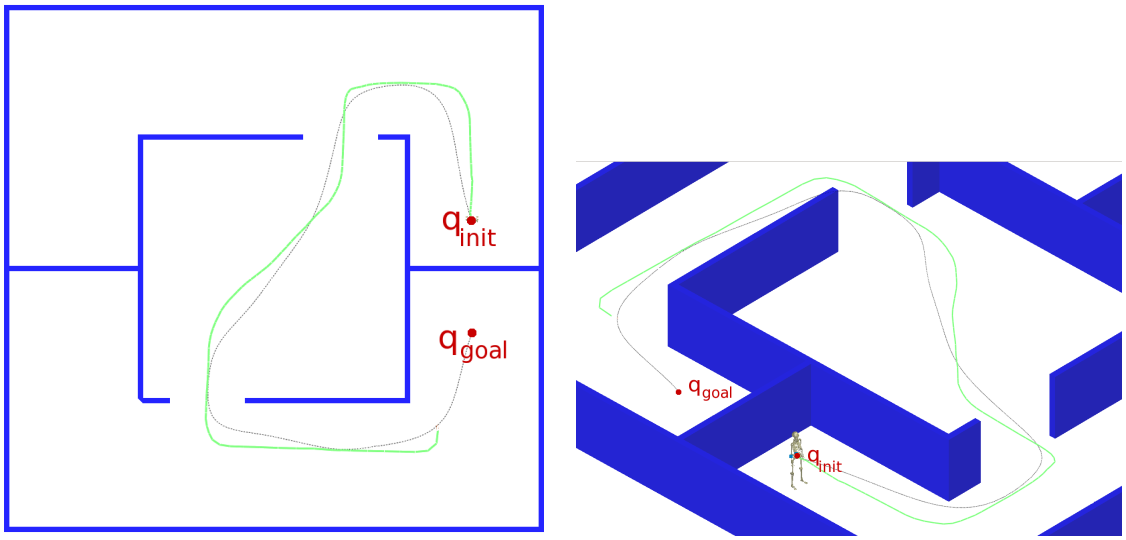
Un environnement simple

Dans cet exemple, illustré figure 4.11, l'utilisateur doit guider une planification entre la partie supérieure de l'environnement et la partie inférieure en passant par la pièce du milieu. L'environnement étant simple et avec peu d'obstacles, il n'a pas de difficulté à guider la planification efficacement. Nous avons donc utilisé une méthode d'échantillonnage autour du chemin de l'utilisateur, sans composante d'échantillonnage uniforme. Sur la figure 4.11-a, on peut voir en vert le chemin emprunté par l'utilisateur pour guider la planification et en noir le graphe construit à partir de ses déplacements. On voit que le graphe est très proche du chemin et que pour la partie finale de la planification, l'utilisateur n'a pas besoin d'aller jusqu'à la configuration finale car l'algorithme de planification la connecte directement dès qu'un de ses nœuds est dans le domaine de visibilité de cette configuration. Sur la figure 4.11-b on peut voir le chemin obtenu après la transformation par courbes de Bézier.



(a) (b)

FIG. 4.11 – (a) En vert, le chemin parcouru par l'avatar sous le contrôle de l'utilisateur, via le périphérique d'interaction. En noir, le graphe construit lors de la planification semi-interactive. (b) Le chemin obtenu après transformation par courbes de Bézier.



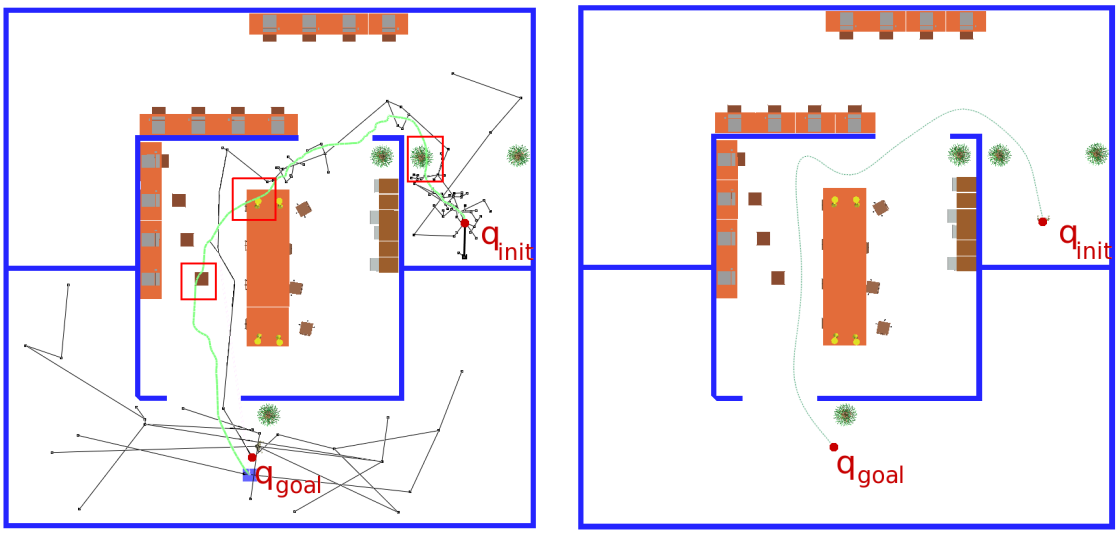
(a) (b)

FIG. 4.12 – (a) En vert, le chemin parcouru par l'avatar sous le contrôle de l'utilisateur. En noir le chemin optimisé. (b) La même scène, d'un point de vue différent.

Les figures 4.12-a et 4.12-b permettent de comparer directement le chemin parcouru par l'utilisateur avec le chemin final. On peut remarquer que le chemin final reste très proche de celui de l'utilisateur tout en étant lisse et réaliste pour une locomotion humaine. Après la transformation par courbe de Bézier, l'aspect en dents de scie dû à la diffusion aléatoire n'apparaît plus et la courbe a un aspect plus naturel. C'est le comportement désiré pour le planificateur de locomotion.

Un environnement encombré

Dans cet exemple, illustré figure 4.13, l'utilisateur guide la planification à travers le même type d'environnement qui contient maintenant plusieurs obstacles. Sur la figure 4.13-a on peut voir en vert le chemin parcouru par l'utilisateur. Celui-ci est approximatif car l'utilisateur manque de dextérité avec le périphérique d'interaction. Certaines parties sont donc en collision avec l'environnement (entourées en rouge sur la figure). Pour cet environnement, nous avons utilisé une méthode d'échantillonnage autour du chemin de l'opérateur avec une composante d'échantillonnage uniforme, afin de pallier le manque de dextérité de l'utilisateur. Sur la même figure, on peut voir en noir le graphe construit lors de la planification. Une partie de ce graphe suit le chemin tracé par l'utilisateur et une autre explore l'environnement indépendamment. On observe que les parties en collision du chemin utilisateur ont été modifiées. Sur la figure 4.13-b, la trajectoire finale du personnage virtuel est affichée. Celle-ci évite les obstacles que l'utilisateur n'a pas pu contourner.

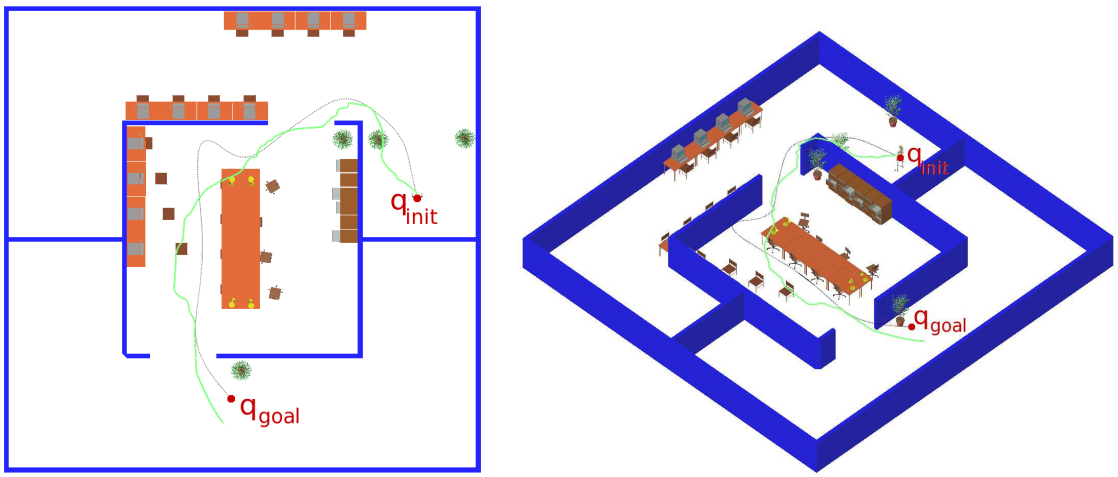


(a) (b)

FIG. 4.13 – (a) En vert, le chemin parcouru par l'avatar sous le contrôle de l'utilisateur, via le périphérique d'interaction. En noir, le graphe construit lors de la planification semi-interactive. (b) Le chemin obtenu après transformation par courbes de Bézier.

Sur la figure 4.14 on peut comparer le chemin de l'utilisateur avec la trajectoire finale de

l'animation. Tout comme pour l'environnement précédent, on observe que la solution reste proche du chemin de l'utilisateur, tout en gardant un aspect naturel et en corrigeant le tracé en collision de celui-ci.



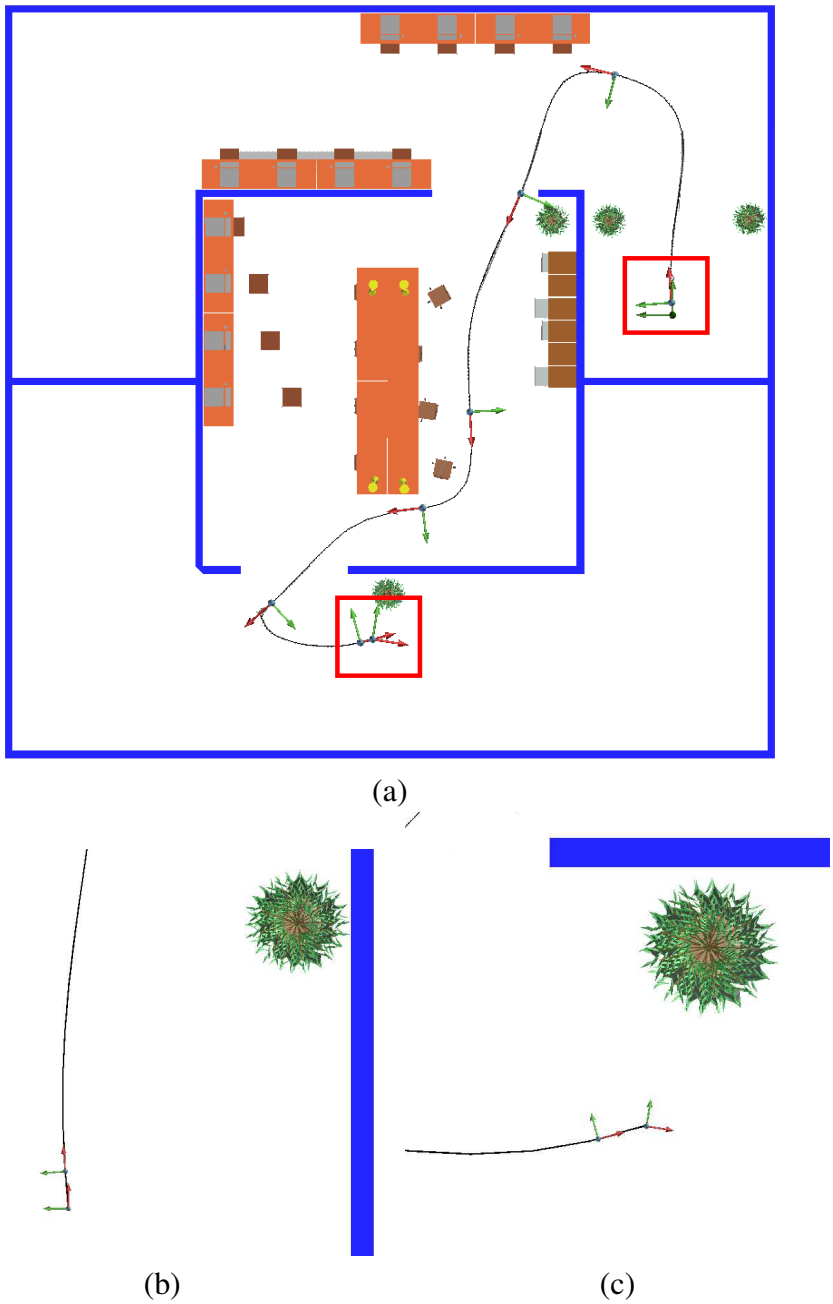
(a) (b)

FIG. 4.14 – (a) En vert, le chemin parcouru par l'avatar sous le contrôle de l'utilisateur. En noir le chemin optimisé. (b) La même scène, d'un point de vue différent.

Une locomotion partiellement holonome

Sur la figure 4.15-a, on peut observer une trajectoire planifiée par la version semi-interactive de l'algorithme. Le long de cette trajectoire, les repères affichés à l'aide de flèches rouges, bleues et vertes représentent la position et l'orientation de certaines configurations de la trajectoire. La flèche rouge pointe vers l'avant du personnage virtuel. On remarque que l'orientation du personnage est tangente à la trajectoire tout le long, excepté pour les configurations initiale et finale. Celles-ci conservent leur orientation initiale qui leur a été spécifiée par l'utilisateur au début de la requête de planification. Cette différence d'orientation par rapport à la courbe peut être mieux observée sur les figures 4.15-b et 4.15-c. La partie de la trajectoire finale qui se situe entre les deux premières et les deux dernières configurations à été interpolée linéairement, contrairement au reste de la courbe (où l'interpolation est faite avec des courbes de Bézier lors de la transformation décrite dans la section 4.2.2).

Le figure 4.16 présente le résultat de l'animation du personnage à l'aide du planificateur semi-interactif.



(a)

(b) (c)

FIG. 4.15 – (a) En noir, une trajectoire obtenue avec le planificateur semi-interactif. Les repères indiquent les positions et orientations de quelques configurations. Les cadres rouges indiquent les parties holonomes de la trajectoire qui sont montrées de plus près dans les figures (b) et (c) ci-contre.

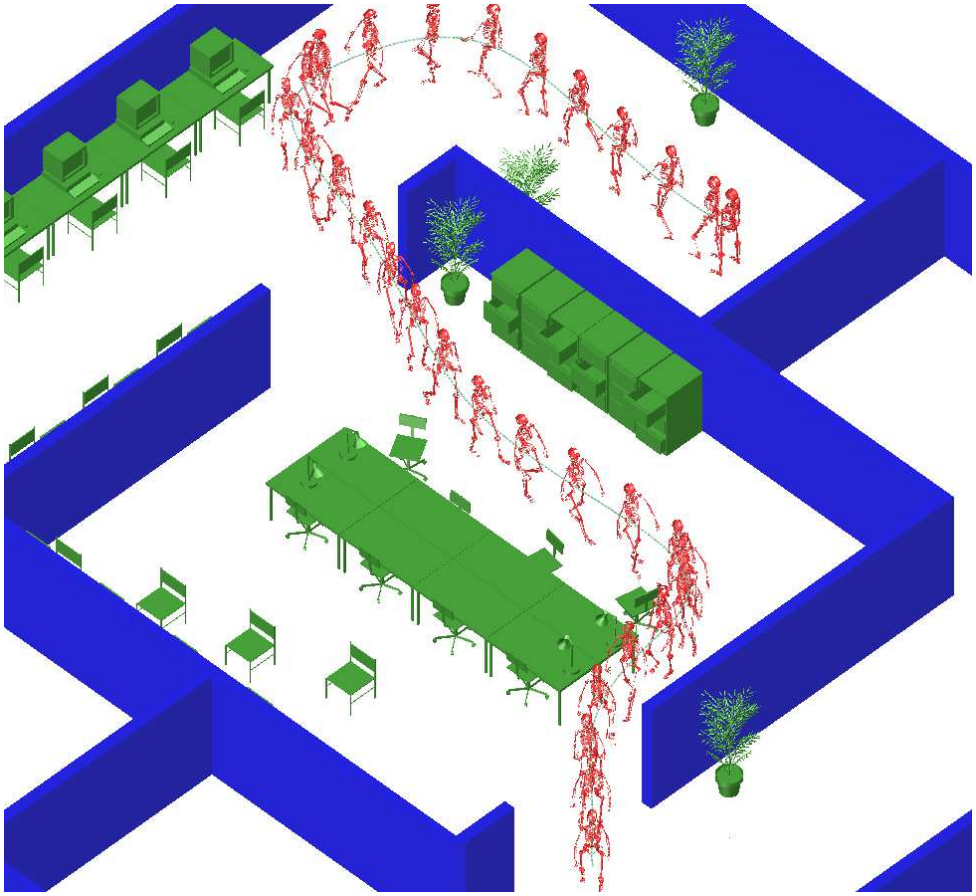


FIG. 4.16 – Illustration de l'animation du personnage le long de la trajectoire.

4.2.3.2 Le planificateur interactif

Pour illustrer le planificateur interactif, nous avons utilisé les mêmes environnements que dans la section précédente. La figure 4.17 présente la trajectoire planifiée et animée à l'aide de l'opérateur et du périphérique interactif à six moments différents. Le temps de planification a été limité à 5 itérations (t_{planif}) du planificateur interactif (*IRRT*, la phase de planification décrite dans la section 4.2.2). Cela permet à l'utilisateur d'avancer suffisamment pour obtenir un chemin assez long pour être animé (ce chemin doit contenir au moins 2 configurations séparées par une distance supérieure ou égale à longueur d'un pas du personnage virtuel). Sur les figures 4.17-a, 4.17-b et 4.17-c la transformation par courbes de Bézier et l'animation n'ont pas encore été faites. Le chemin construit suit le déplacement de l'utilisateur. Sur la figure 4.17-d, après la première itération, la première partie du chemin a été optimisée puis animée et le chemin parcouru par l'utilisateur depuis la fin de la première itération a été ajouté. Les figure 4.17-e et 4.17-f montrent la suite de la seconde itération.

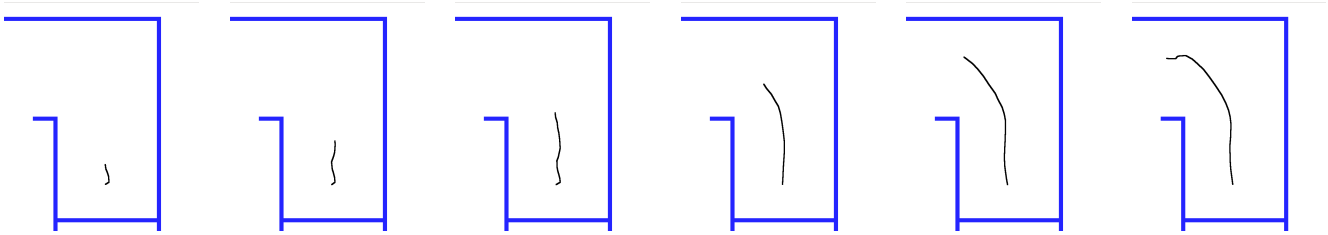
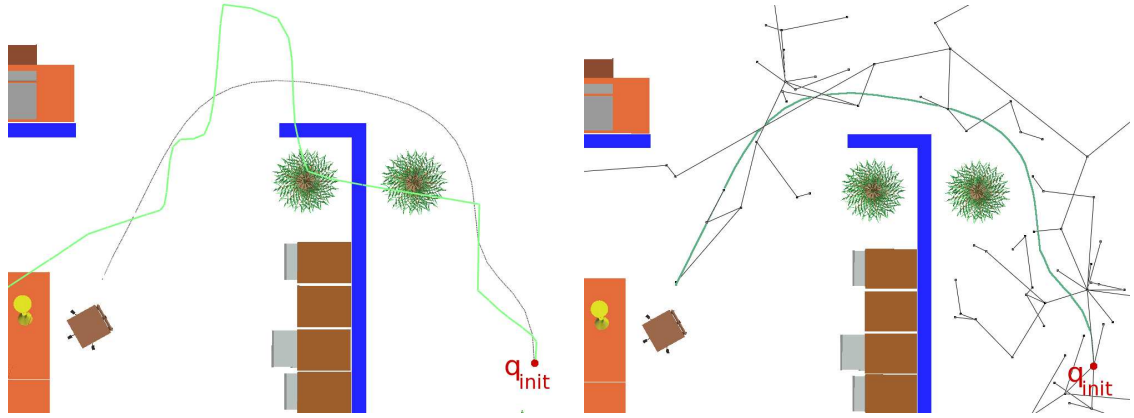


FIG. 4.17 – Différentes itérations du planificateur interactif, par ordre chronologique, dans le même environnement que celui de la figure 4.11

Sur la figure 4.18 on peut voir un exemple d'exécution du planificateur interactif dans l'environnement encombré. Sur la figure de gauche on voit que l'utilisateur maîtrise mal le périphérique d'interaction, et effectue un guidage approximatif de la planification. La méthode *ILP* interactive parvient tout de même à contourner les obstacles et à suivre l'utilisateur. Ici la méthode d'échantillonnage ne possède pas de composante uniforme car on veut privilégier le contrôle de l'utilisateur sur la résolution du problème. Nous avons donc utilisé un échantillonnage autour du chemin de l'utilisateur, en augmentant le rayon d'influence du chemin (l'écart type, voir chapitre 3). Ici, t_{planif} a été fixé à 10 itérations.

4.3 Conclusion et Perspectives

Dans ce chapitre, nous avons décrit un planificateur de locomotion interactif permettant de combiner un algorithme de planification de mouvement et un contrôleur de locomotion pour



(a) (b)

FIG. 4.18 – (a) En vert, une trajectoire utilisateur en collision. En noir, la trajectoire obtenue pendant le déplacement de l'utilisateur avec le planificateur interactif. (b) En noir, on peut voir le graphe construit par le planificateur interactif lors de la planification.

faire marcher un personnage virtuel. Cette méthode repose sur l'algorithme de planification interactive décrit dans le chapitre 3 et un contrôleur de locomotion basé sur l'interpolation de captures de mouvement. Cette méthode nommée *ILP* a été implémentée en deux versions : une version semi-interactive et une version interactive. Le *ILP* semi-interactif découple la partie planification de mouvement interactive de la partie animation. Le chemin qui sera parcouru par le personnage virtuel est obtenu grâce au *IRRT*, ce qui lui permet d'être conforme aux critères de l'utilisateur tout en étant réaliste grâce à la méthode de transformation basée sur les courbes de Bézier utilisée ou à la méthode locale utilisant les courbes de Bézier décrite. Cette version semi-interactive peut permettre, par exemple, dans un contexte d'animation d'éditer une trajectoire de manière simple, intuitive et rapide en guidant uniquement la direction de la marche vers le but. Ainsi l'évitement des obstacles est fait automatiquement, mais la trajectoire suit le chemin désiré par l'opérateur, qui n'est pas forcément celui qu'un planificateur automatique aurait trouvé.

Le *ILP* interactif permet un contrôle augmenté du personnage virtuel, dans la mesure où le contrôle exercé par l'opérateur grâce au périphérique interactif est corrigé par l'algorithme de planification de mouvement pour éviter les collisions et optimiser une trajectoire utilisateur qui peut parfois être approximative. La méthode de transformation par courbes de Bézier développé dans le cadre du *ILP* interactif permet d'intégrer les aspects holonomes de la marche humaine au début et à la fin des trajectoires animées (lorsque le personnage virtuel se déplace juste avant ou juste après un temps d'arrêt). La modification des différents paramètres utilisés pour le planificateur (limite de temps pour une itération et nombre de configurations affichées à chaque itération) permet d'avoir un chemin plus ou moins naturel globalement en lissant une portion de trajectoire plus ou moins grande. En effet à chaque itération la trajectoire n'étant lissée que

localement (sur la portion planifiée durant la même itération) la trajectoire globale peut être assez sinueuse, en fonction du chemin parcouru par l'utilisateur.

Perspectives

Il serait intéressant d'étudier les effets de la variations de t_{planif} sur l'aspect de la trajectoire finale, ainsi que sur les performances de la méthode en terme d'interactivité. Si le temps est trop important, une itération deviendrait alors trop longue et le déplacement de l'utilisateur et du personnage virtuel à l'écran seraient de plus en plus saccadés, principalement à cause du calcul de l'animation. De plus, l'effet de différentes méthodes d'échantillonnage sur les performances des deux types de planificateur semble intéressant à étudier. Une méthode avec une composante uniforme donnerait des trajectoires globales plus sinueuses et plus indépendantes du guidage de l'utilisateur, ce qui n'est pas nécessairement voulu dans le cadre de l'animation d'un personnage. Il serait intéressant de développer une méthode utilisant une composante uniforme et permettant de contourner ce problème. Il faut remarquer que dans le cas du planificateur semi-interactif la trajectoire finale est une courbe sans boucle car un déplacement circulaire de l'utilisateur entrainerait automatiquement la création d'une nouvelle branche dans l'arbre à partir du nœud le plus proche (car la méthode de base est le *RRT*). Dans certaines applications, on peut vouloir faire tourner le personnage autour d'un obstacle (bâtiment, table). Dans ces cas, la méthode utilisée pourrait être adaptée pour accepter des trajectoires contenant des boucles.

5 Conclusion

5.1 Conclusion

Dans le premier chapitre, nous avons développé une méthode de planification de mouvement pour un type d'environnement spécifique contenant une ou plusieurs zones séparées par des passages étroits, type d'environnement que nous avons appelé environnements faiblement connexes. En partant de méthodes de planification de mouvement probabilistes existantes, nous avons combiné deux approches différentes en une seule méthode qui exploite les avantages de chacune pour remédier aux inconvénients de l'autre.

Les méthodes sur lesquelles nous nous sommes basés sont d'une part la méthode des graphes probabilistes de visibilité ($VISPRM$) pour le nombre de nœuds réduit qui sont ajoutés au graphe construit et d'autre part la méthode des arbres locaux ($LTRRT$) pour sa capacité d'exploration de zones difficilement accessibles de l'environnement. La nouvelle méthode résultant de cette combinaison est appelée « Arbres locaux de visibilité » ($VISLT$). Nous sommes parvenus à combiner ces deux méthodes en introduisant la notion de « nœuds éclaireurs » qui permettent d'améliorer la visibilité commune entre deux composantes connexes séparées par un passage étroit en se rapprochant de ce passage. Un coefficient d'extension permet de contrôler l'ajout de ces nœuds et ainsi de conserver l'avantage d'un nombre de nœuds réduit tout en favorisant l'exploration des zones difficiles d'accès.

De nombreux tests et l'analyse de leurs résultats ont permis de montrer la validité de notre approche pour le type d'environnements considéré ainsi que de meilleures performances pour

l'approche combinée que pour les deux approches séparées.

Dans le second chapitre, nous avons développé une méthode de planification de mouvement permettant de faire coopérer un opérateur humain et un algorithme de planification de mouvement probabiliste afin de résoudre des problèmes de planification. Cette méthode se base sur une interaction naturelle et continue entre l'opérateur et l'algorithme via une scène virtuelle et à l'aide de périphériques tels qu'une souris 6D ou un bras haptique. À la différence d'autres méthodes faisant intervenir un opérateur humain, notre méthode permet l'échange continu d'informations entre opérateur et algorithme dans une seule boucle d'interaction au lieu de décomposer l'intervention de chacun en plusieurs étapes distinctes.

Le principe repose sur la déformation de la zone d'échantillonnage aléatoire des configurations pour prendre en compte les actions de l'utilisateur et les informations récoltées par l'algorithme tout au long de son exploration de l'environnement. Chaque partie profite de l'expérience de l'autre pour accélérer la recherche de solution. Plusieurs paramètres permettent de modifier l'influence de l'opérateur sur la recherche en fonction du problème et du type de solution recherchée et de résoudre plusieurs problèmes liés à la dextérité de l'utilisateur et à la complexité du problème.

Une étude de la variation des paramètres les plus significatifs pour l'interaction a permis de montrer les différents avantages de cette méthode en fonction de la nature de l'environnement (encombré ou vide, avec ou sans passages étroits) et du but recherché par la coopération entre les deux parties (privilégier le contrôle de l'utilisateur pour avoir une solution plus naturelle - un chemin lisse - ou bien privilégier la résolution du problème, ce qui implique une solution qui pourra être plus en « ligne brisée »).

Enfin, dans le dernier chapitre, nous avons intégré notre méthode de planification interactive avec un contrôleur de locomotion pour permettre l'animation d'un personnage virtuel. Le planificateur de locomotion interactif qui en découle permet de faciliter la création de trajectoires de marche sans collisions dans des environnements encombrés pour un personnage. En se basant sur des méthodes d'interpolation de mouvements capturés, le planificateur automatise la création et l'animation de la trajectoire tout en respectant l'intention de l'utilisateur. Le planificateur interactif développé au chapitre précédent permet de garantir le respect du chemin donné en entrée par l'utilisateur, tout en évitant les collisions avec les obstacles de la scène virtuelle qui peuvent être dues à un manque de dextérité de l'utilisateur.

Le planificateur à été développé en deux versions. La première version, dite semi-interactive, est plus orientée pour la recherche d'une trajectoire sans collisions d'aspect naturel pour un personnage virtuel. Cette version se focalise sur la découverte d'une solution en exploitant l'aide de l'utilisateur. La seconde version favorise un contrôle plus important de l'utilisateur tout en permettant l'évitement des collisions dues à son manque d'expérience avec le périphérique

d'interaction. La planification et l'animation sont faites au fur et à mesure que l'utilisateur se déplace. Dans les deux versions, la trajectoire planifiée intègre l'aspect holonome de la marche humaine au début et à la fin de la trajectoire (le personnage virtuel effectue un pas de manière holonome, de façon à se placer dans la direction de déplacement).

Plusieurs exemples viennent illustrer les deux versions du planificateur en soulignant le respect de la trajectoire désirée par l'opérateur et l'évitement automatique des collisions.

5.2 Perspectives

En ce qui concerne la planification de mouvement dans des environnements faiblement connexes, la perspective de pouvoir modifier dynamiquement le coefficient d'extension d'un arbre local pourrait améliorer les performances de l'algorithme. En effet, un coefficient élevé au début de la diffusion de l'arbre (lorsqu'il y a peu de nœuds) permettrait une exploration de la zone plus rapide et plus extensive afin de trouver les passages étroits. La diminution de ce coefficient en fonction des échecs des tentatives de connexion avec d'autres arbres permettrait de densifier l'exploration de la scène aux abords de la zone pour tenter de trouver les passages les plus difficiles. L'affectation d'un coefficient unique pour chaque arbre local créé permettrait de mieux contrôler la diffusion globale du graphe en prenant en compte les informations de position relative de chaque arbre. Pour deux arbres qui ont une forte probabilité de pouvoir être connectés ensemble à un moment donné l'algorithme modifierait le coefficient d'extension pour permettre une connexion plus rapide.

En ce qui concerne la planification de mouvement interactive, la modélisation des contacts et des glissements dans la scène virtuelle permettrait un guidage plus précis de l'algorithme par l'utilisateur. En effet, lors de certains problèmes (assemblage, manipulation) les contacts et les glissements peuvent servir de guide à l'utilisateur lorsqu'il déplace l'objet, notamment dans des zones difficilement visualisables. Ces informations de contact augmenteraient la précision de la trajectoire de l'utilisateur et permettraient par conséquent un meilleur guidage de la recherche de chemin. L'ajout de ces informations peut se faire par l'intermédiaire d'un retour haptique ou uniquement dans la scène virtuelle. L'utilisateur n'aurait alors qu'un retour visuel des contacts et des glissements.

L'utilisation de plus d'un périphérique d'interaction lors d'une recherche de chemins offrirait plusieurs avantages. Le premier serait de pouvoir explorer plusieurs pistes en même temps lors d'une même requête de planification interactive. Les différents périphériques d'interaction, pilotés par un seul opérateur ou par plusieurs seraient représentés par autant d'avatars dans la scène virtuelle. La méthode d'échantillonnage permettrait donc d'obtenir des échantillons en suivant la position de ces différents avatars. L'échantillonnage peut se faire alternativement autour d'une position et de l'autre (ainsi qu'à l'aide d'une composante uniforme

si cela améliore les performances de la recherche) ou bien en parallèle dans différents processus. La visualisation peut se faire sur le même écran pour des problèmes simples ou sur des écrans différents pour que chaque utilisateur ait le point de vue le plus avantageux pour l'avatar qu'il manipule. Les informations récoltées par chaque opérateur (collisions, zones préférentielles) seraient retournées à tous les utilisateurs.

Le second avantage serait de pouvoir déplacer plusieurs corps dans la scène virtuelle nécessitant une coordination entre eux. Par exemple, le déplacement d'un meuble par deux personnages virtuels pourrait bénéficier de trois périphériques d'interaction : un pour chaque personnage et un pour la table. Les configurations dans ce problème contiendraient les paramètres pour chaque personnage et pour la table. Enfin, il serait possible d'effectuer une recherche de chemin pour un objet déformable plus efficacement avec deux périphériques. La présence d'un second périphérique ferait bénéficier l'utilisateur d'un second point de saisie lui permettant de déformer l'objet à sa guise pour le faire entrer dans des passages complexes. Le pliage de molécules est un exemple d'application pouvant bénéficier de plusieurs périphériques.

En ce qui concerne l'animation de personnages virtuels à l'aide d'une méthode de planification interactive, une intégration plus complète de l'aspect holonome de la marche humaine pourrait être considérée. En effet dans l'implémentation actuelle cet aspect ne se retrouve qu'aux extrémités des trajectoires. Or, lors d'un déplacement naturel, une portion de trajectoire holonome peut apparaître par exemple lorsque le personnage évite un petit obstacle qui se trouve sur sa route (un piéton modifie sa trajectoire pour éviter un lampadaire en marchant de biais). Un autre exemple peut être le passage dans un couloir ou une ouverture trop étroits pour une marche avant.

L'utilisation d'autres bibliothèques de capture de mouvement pour avoir des modes de locomotion différents peut être envisagée : locomotion à quatre pattes ou en sautillant (à la manière d'un boxeur sur un ring) par exemple. La planification et l'animation de la trajectoire se faisant de manière automatique, seuls la prise de captures et le pré-traitement de celles-ci sont nécessaires pour effectuer ce changement de bibliothèque. Une utilisation de manière indépendante de chaque bibliothèque peut être envisagée dans un premier temps puis une utilisation mixte ensuite. Cela nécessiterait d'une part le calcul d'une animation de transition entre deux types de locomotion, et d'autre part un moyen de déclencher ce changement de locomotion. Ce moyen peut être soit interactif - l'utilisateur appuie sur une touche ou un bouton du périphérique interactif - soit automatique, en utilisant par exemple les graphes de mouvements introduits dans [Kovar et al. 2008]. Cette méthode combinant différents modes de locomotion permettrait d'intégrer le déplacement du personnage sur des plans inclinés ou sur des escaliers.

La méthode ILP permet l'évitement d'obstacles en modifiant la trajectoire du personnage, mais il peut parfois être préférable de ne pas modifier cette trajectoire (éviter une branche ou

un tuyau en hauteur) et de ne déplacer que la partie supérieure du corps. Cet évitement local d'obstacle à déjà été étudié dans les travaux de [Pettré 2003], mais nous ne l'avons pas utilisé ici pour des raisons de performances. La détection de collisions sur la trajectoire animée et la déformation de la trajectoire en conséquence pourraient être intégrées dans le planificateur de locomotion interactif en étudiant l'impact sur les performances et l'interactivité. De plus, dans ces travaux cet évitement ne concerne que les obstacles que l'on peut éviter sans modifier la position des jambes. Dans ce dernier cas, il serait intéressant de pouvoir modifier la trajectoire en interpolant entre deux modes de locomotion différents (et donc en faisant appel à deux bibliothèques de captures différentes). Par exemple, pour passer sous une branche un peu basse, la trajectoire serait modifiée en interpolant entre une locomotion de type « marche avant » et une locomotion de type « marche accroupie ». De même pour enjamber un obstacle l'utilisation des types de locomotion « marche avant », « montée d'escalier » et « descente d'escalier » pourraient être utilisés.

Enfin, de manière plus générale, l'ajout de la dynamique pourrait être considérée pour avoir des animations plus réalistes et qui interagiraient mieux avec l'environnement. L'ajout de contraintes dynamiques permettrait de déformer les mouvements interpolés préalablement en fonction des forces de contact, de gravité et d'inertie.

6
Références

Références

AMATO, N. M., BAYAZIT, O. B., DALE, L. K., JONES, C., AND VALLEJO, D. 1998. OBPRM : An obstacle-based PRM for 3D workspaces. In *Workshop on Algorithmic Foundations of Robotics*. 40

ARIKAN, O. AND FORSYTH, D. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics 21,* 3, 483–490. 76

BARBE, L., BAYLE, B., GANGLOFF, J., DE MATHELIN, M., AND PICCIN, O. 2007. Design and evaluation of a linear haptic device. In *Robotics and Automation, 2007 IEEE International Conference on*. 485 –490. 42

BARRAQUAND, J., KAVRAKI, L., LATOMBE, J., MOTWANI, R., LI, T., AND RAGHAVAN, P. 1997. A random sampling scheme for path planning. *The International Journal of Robotics Research 16,* 6, 759. 49

BAUMANN, R. AND CLAVEL, R. 1997. Haptic interface for virtual reality based laparoscopic surgery training environment. *Unpublished doctoral dissertation 1734*. 43

BAYART, B. AND KHEDDAR, A. 2007. Evaluation of an haptic step-guidance algorithm through a teaching 2d/3d path scenario. In *Haptic, Audio and Visual Environments and Games, 2007. HAVE 2007. IEEE International Workshop on*. 124 –129. 43

BAYART, B., POCHEVILLE, A., AND KHEDDAR, A. 2005. An adaptive haptic guidance software module for i-touch : example through a handwriting teaching simulation and a 3d maze. In *Haptic Audio Visual Environments and their Applications, 2005. IEEE International Workshop on*. 6 pp. 43

BAYAZIT, O. B., SONG, G., AND AMATO, N. M. 2000. Enhancing randomized motion planners : Exploring with haptic hints. In *Int. Conference on Robotics and Automation (ICRA'2000)*. 529–536. 41

BAYAZIT, O. B., SONG, G., AND AMATO, N. M. 2001. Ligand binding with obprm and haptic user input. In *Int. Conference on Robotics and Automation (ICRA'2001)*. 42

BOOR, V., OVERMARS, M. H., AND VAN DER STAPPEN, A. F. 1999. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings IEEE International Conference on Robotics & Automation*. 1018–1023. 40

BOTTESI, G., LAUMOND, J., AND FLEURY, S. 2004. A Motion Planning Based Video Game. 75

BOULIC, R. 2005. Proactive steering toward oriented targets. *Eurographics Short presentation program (to appear)*. 76

BOULIC, R., DONIKIAN, S., AND MULTON, F. 2006. Le traité de la Réalité Virtuelle, volume 3, chapter Modeles pour les humanoıdes. *Mines Paris,*, 273–285. 39

BRUDERLIN, A. AND WILLIAMS, L. 1995. Motion signal processing. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 104. 75

BURDEA, G. 2000. Haptics issues in virtual environments. In *Computer Graphics International*. 295–302. 39

BURTNYK, N. AND WEIN, M. 1971. Computer generated key frame animation. *Journal of the Society of Motion Picture and Television Engineers 80,* 3, 149–153. 75

CANNY, J. 1988. *The complexity of robot motion planning*. The MIT Press. 10

CHABAL, C., MEGARD, C., AND SIBILE, L. 2005. Emm-3d : a virtual environment for evaluating maintenability from cad models. In *Laval Virtual 2005*. 43

CHEN, L. AND BROWN, C. G. 2005. A 3d mouse for interacting with virtual objects. In *IEEE International Symposium on Circuits and Systems*. 40

CHOI, M., LEE, J., AND SHIN, S. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics (TOG) 22,* 2, 203. 76

CHOSET, H., LYNCH, K., HUTCHINSON, S., KANTOR, G., BURGARD, W., KAVRAKI, L., AND THRUN, S. 2005. *Principles of Robot Motion : theory, algorithms, and implementation*. MIT Press. 9

DALIBARD, S. AND LAUMOND, J. 2009. Control of probabilistic diffusion in motion planning. *Algorithmic Foundation of Robotics VIII*, 467–481. 40

DELAUNAY, B. 1934. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7*, 793–800. 79

DURIEZ, C., DUBOIS, F., KHEDDAR, A., AND ANDRIOT, C. 2006. Realistic haptic rendering of interacting deformable objects in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on 12,* 1 (jan.-feb.), 36 –47. 42

ESTEVES JARAMILLO, C. 2007. Motion planning : from digital actors to humanoid robots. Ph.D. thesis, Institut National Polytechnique, Toulouse, 100p. Doctorat. 7, 8, 11, 16

FERRÉ, E. AND LAUMOND, J. 2004. An iterative diffusion algorithm for part disassembly. In *Int. Conference on Robotics and Automation (ICRA'2004)*. 3149–3154. 40, 61

FERRÉ, E., LAUMOND, J., ARECHAVALETA, G., AND ESTEVÉS, C. 2005. Progresses in assembly path planning. In *Int. Conference on Product Lifecycle Management (PLM'05)*. 373–382. 40

FUCHS, P. 2006. Le traité de la réalité virtuelle, Vol. 2 : L'interfaçage, l'immersion & l'interaction en environnement virtuel. 39

FUCHS, P. AND DONIKIAN, S. 2009. *Le traité de la réalité virtuelle : Volume 5 : Les humains virtuels*. TRANSVALOR Presses des MINES. 76

GALEANO, D. AND PAYANDEH, S. 2005. Artificial and natural force constraints in haptic-aided path planning. In *Int. Workshoop on Haptic Audio Visual Environnments and their Applications*. 45–50. 40

GLARDON, P. 2006. On-line locomotion synthesis for virtual humans. Ph.D. thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE. 76

HASTINGS, W. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika 57,* 1, 97–109. 49

HE, X. AND CHEN, Y. 2009. Haptic-aided robot path planning based on virtual tele-operation. *Robot. Comput.-Integr. Manuf. 25,* 4-5, 792–803. 42

KANEHIRO, F., SULEIMAN, W., LAMIRAUX, F., YOSHIDA, E., AND LAUMOND, J. 2008. Integrating Dynamics into Motion Planning for Humanoid Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*. 660–667. 73

KAVRAKI, L. E., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation 12,* 4 (June), 566–580. 10

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2008. Motion graphs. In *ACM SIGGRAPH 2008 classes*. ACM, 51. 75, 102

KUFFNER, J. 2004. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation*. Vol. 4. Citeseer, 3993–3998. 49

KUFFNER, J.J., J. AND LAVALLE, S. 2000. Rrt-connect : An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. Vol. 2. 995 –1001 vol.2. 13

LADEVEZE, N., FOURQUET, J., PUEL, B., AND TAÏX, M. 2009. Haptic assembly and disassembly task assistance using interactive path planning. In *IEEE Virtual Reality (IEEE VR 09)*. 42

LADEZEVE, N., FOURQUET, J. Y., AND TAÏX, M. 2008. Interactive motion planning in virtual reality environments. In *Virtual Reality International Conference (VRIC'08)*. 42

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Kluwer, Boston, MA. 9

LAU, M. AND KUFFNER, J. 2005. Behavior planning for character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 280. 76

LAUMOND, J. 2006. Kineo cam : a success story of motion planning algorithms. *IEEE Robotics & Automation Magazine 13,* 2 (june), 90–93. 40

LaValle, S. and Kuffner, J.J., J. 1999. Randomized kinodynamic planning. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 1. 473 –479 vol.1. 13

LaValle, S. M. 1998. Rapidly-exploring random trees : A new tool for path planning. Tech. Rep. 98-11, Computer Science Dept., Iowa State University. Oct. 12

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http ://planning.cs.uiuc.edu/. 13

Lecuyer, A., Kheddar, A., Coquillart, S., Graux, L., and Coiffet, P. 2001. A haptic prototype for the simulations of aeronautics mounting/unmounting operations. In *Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on*. 182 –187. 43

Lumelsky, V. and Stepanov, A. 1987. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica 2,* 1, 403–430. 75

Mombaur, K., Truong, A., and Laumond, J.-P. 2010. From human to humanoid locomotion–an inverse optimal control approach. *Autonomous Robots 28,* 3, 369–383. 76

Moser, L. 1966. Moving Furniture Through a Hallway. *SIAM Review 8*, 381–381. 7

Pettré, J. 2003. Planification de mouvements de marche pour acteurs digitaux. Ph.D. thesis. Doctorat. 76, 78, 79, 80, 81, 82, 103

Rosell, J., Vazquez, C., Perez, A., and P.Iniguez. 2008. Motion planning for haptic guidance. *Journal of Intelligents. Robotics Systems 53,* 3, 223–245. 42

Salomon, B., Garber, M., Lin, M., and Manocha, D. 2003. Interactive navigation in complex environments using path planning. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM, 50. 75

Schwartz, J. and Sharir, M. 1987. On the piano mover's problem : III. *Planning, Geometry, and Complexity of Robot Motion, edited by JT Schwartz, M. Sharir, and J. Hopcroft, Ablex, New York*. 7

Shiller, Z., Yamane, K., and Nakamura, Y. 2001. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*. Vol. 1. IEEE ; 1999, 1–8. 75

Siméon, T., Chatila, R., and Laumond, J. 2002. Computer aided motion for logistics in nuclear plants. In *Int. Symposium on Artificial Intelligence, Robotics and Human Centered Technology for Nuclear Applications (AIR'02)*. 46–53. 40

Siméon, T., Laumond, J.-P., Geem, C. V., and Cortés, J. 2001. Computer aided motion : Move3d within molog. In *Int. Conference on Robotics and Automation (ICRA'2001)*. 1494–1499. 40

Siméon, T., Laumond, J.-P., and Nissoux, C. 2000. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics 14,* 6. 15

SRENG, J., LÉCUYER, A., MÉGARD, C., AND ANDRIOT, C. 2006. Using visual cues of contact to improve interactive manipulation of virtual objects in industrial assembly/maintenance simulations. *IEEE Transactions on Visualization and Computer Graphics 12*, 1013–1020. 72

STRANDBERG, M. 2004. Augmenting RRT-planners with local trees. In *Proceedings IEEE International Conference on Robotics & Automation*. 3258–3262. 14, 15, 40

STURMAN, D. 1984. Interactive keyframe animation of 3-D articulated models. In *National Computer Graphics Association of Canada Conference– Graphics Interface'84*. 35–40. 75

SU, Y., ZHU, W., AND YU, T. 2008. Virtual assembly platform based on pc. In *International Conference on Audio, Language and Image*. 40

SUD, A., ANDERSEN, E., CURTIS, S., LIN, M., AND MANOCHA, D. 2007. Real-time path planning for virtual agents in dynamic environments. In *IEEE Virtual Reality Conference, 2007. VR'07*. 91–98. 76

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*. ACM, 106. 76

SUN, Z., HSU, D., JIANG, T., KURNIAWATI, H., AND REIF, J. H. 2005. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics 21,* 6, 1105–1115. 40

TRUONG, A. 2010. Unifying nonholonomic and holonomic behaviors in human locomotion. Ph.D. thesis, Institut National Polytechnique, Toulouse. Doctorat. 89

TRUONG, T. V. A., FLAVIGNÉ, D., PETTRÉ, J., MOMBAUR, K., AND LAUMOND, J.-P. 2010. Reactive synthesizing of human locomotion combining nonholonomic and holonomic behaviors. *Proceedings of the 3rd Biennial IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics (Biorob 2010)*. 76, 80, 81, 82

VAZQUEZ, C. AND ROSELL, J. 2007. Haptic guidance based on harmonic functions for the execution of teleoperated assembly tasks. In *IFAC Workshop on Intelligent Assembly and Disassembly*. 42

WILMARTH, S., AMATO, N., AND STILLER, P. 1999a. MAPRM : A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*. Citeseer, 1024–1031. 49

WILMARTH, S. A., AMATO, N. M., AND STILLER, P. F. 1999b. Maprm : A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceeding of IEEE International Conference on Robotics & Automation*. 40

YE, N., BANERJEE, P., BANERJEE, A., AND DECH, F. 1999. A comparative study of assembly planning in traditional and virtual environments. *IEEE Transactions on Systems, Man, and Cybernetics 29,* 4, 546 –555. 43

YOSHIDA, E., POIRIER, M., LAUMOND, J., KANOUN, O., LAMIRAUX, F., ALAMI, R., AND YOKOI, K. 2008. Whole-body motion planning for pivoting based manipulation by humanoids. In *Proceedings of 2008 IEEE International Conference on Robotics and Automation*. 1712–1717. 73

ZELTZER, D. 1982. Motor control techniques for figure animation. *Computer Graphics and Applications, IEEE 2,* 9 (nov.), 53 –59. 75