THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Informatique et Robotique

Présentée et soutenue par :
Sébastien Dalibard

le : vendredi 22 juillet 2011

Titre :

Planification de mouvement pour systèmes anthropomorphes

JURY
Jean-Paul Laumond
Jean Ponce
James Kuffner
Florent Lamiraux
Etienne Ferré

Ecole doctorale :
Systèmes (EDSYS)

Unité de recherche :
LAAS-CNRS

Directeur(s) de Thèse :

Jean-Paul Laumond

Rapporteurs :
Jean Ponce
James Kuffner

Abstract

This thesis deals with the development and analysis of motion planning algorithms for high dimensional systems : humanoid robots and digital actors. Several adaptations of generic randomized motion planning methods are proposed and discussed.

A first contribution concerns the use of linear dimensionality reduction techniques to speed up sampling algorithms. This method identifies on line when a planning process goes through a narrow passage of some configuration space, and adapts the exploration accordingly. This algorithm is particularly suited to difficult problems of motion planning for computer animation.

The second contribution is the development of randomized algorithms for motion planning under constraints. It consists in the integration of prioritized inverse kinematics tools within randomized motion planning. We demonstrate the use of this method on different manipulation planning problems for humanoid robots. This contribution is generalized to whole-body motion planning with locomotion.

The last contribution of this thesis is the use of previous methods to solve complex manipulation tasks by humanoid robots. More specifically, we present a formalism that represents information specific to a manipulated object usable by a motion planner. This formalism is presented under the name of "documented object".

Résumé

L'objet de cette thèse est le développement et l'étude des algorithmes de planification de mouvement pour les systèmes hautement dimensionnés que sont les robots humanoïdes et les acteurs virtuels. Plusieurs adaptations des méthodes génériques de planification de mouvement randomisées sont proposées et discutées.

Une première contribution concerne l'utilisation de techniques de réduction de dimension linéaire pour accélérer les algorithmes d'échantillonnage. Cette méthode permet d'identifier en ligne quand un processus de planification passe par un passage étroit de l'espace des configurations et adapte l'exploration en fonction. Cet algorithme convient particulièrement bien aux problèmes difficiles de la planification de mouvement pour l'animation graphique.

La deuxième contribution est le développement d'algorithmes randomisés de planification sous contraintes. Il s'agit d'une intégration d'outils de cinématique inverse hiérarchisée aux algorithmes de planification de mouvement randomisés. On illustre cette méthodes sur différents problèmes de manipulation pour robots humanoïdes. Cette contribution est généralisée à la planification de mouvements corps-complet nécessitant de la marche.

La dernière contribution présentée dans cette thèse est l'utilisation des méthodes précédentes pour résoudre des tâches de manipulation complexes par un robot humanoïde. Nous présentons en particulier un formalisme destiné à représenter les informations propres à l'objet manipulé utilisables par un planificateur de mouvement. Ce formalisme est présenté sous le nom d'« objets documentés ».

Remerciements

Je tiens en premier lieu à exprimer toute ma gratitude à mon directeur de thèse Jean-Paul Laumond. La confiance qu'il m'a portée et l'autonomie dont j'ai profité pendant ma thèse furent à l'origine du plaisir que j'ai eu à travailler au LAAS. Je lui suis par ailleurs extrêmement reconnaissant de sa disponibilité en cas de besoin ; j'ai beaucoup appris de ses conseils comme de ses critiques.

Je remercie chaleureusement James Kuffner et Jean Ponce, qui ont accepté d'être rapporteurs de cette thèse. Je suis très honoré du sérieux qu'ils ont mis dans ce travail et de l'intérêt qu'ils ont porté à mes travaux.

Je remercie également Étienne Ferré d'avoir accepté de faire partie de mon jury de thèse.

J'adresse tous mes remerciements à Florent Lamiraux, qui a toujours pris le temps de répondre à mes questions et m'orienter dans mes recherches. Sa connaissance approfondie des problèmes robotiques m'a été d'une grande aide, et je suis très heureux de le compter parmi les membres de mon jury.

Je voudrais profiter de cette occasion pour remercier tous les membres du groupe RIA pour l'atmosphère accueillante et stimulante qui y règne.

J'ai eu la chance de côtoyer et de travailler avec des doctorants sympathiques et brillants pendant ces quelques années au LAAS. La bonne humeur dans laquelle se sont déroulées ces recherches a parfois eu tendance à faire s'estomper la distinction entre étude et récréation. Je commencerai par remercier ceux avec qui j'ai eu la joie de collaborer scientifiquement. Je leur souhaite que ces travaux en commun leur aient été aussi agréables et enrichissants qu'à moi. Un grand merci donc à Alireza Nakhaei, Antonio El Khoury, Oussama Kanoun et Jesse Himmelstein.

Un mot pour mes glorieux aînés gepettistes : je garde d'excellents souvenirs de nos journées au labo comme de nos nombreuses sorties. Merci à Mathieu, Minh, Anh, Manish, David et Francisco. Merci aussi à ceux qui restent d'avoir fait perdurer cette bonne ambiance, et bon courage pour leur fin de thèse à Layale, Sovan, Thomas et Duong.

Je remercie mes colocataires et amis de Toulouse, Camille et Jonathan. C'est en grande partie de leur faute si ces années sont passées trop vite. Merci également à Fabrice, Toulousain assimilé, que ses visites chez nous ont conduit jusqu'à ma soutenance.

Je souhaite enfin exprimer toute ma reconnaissance à mes parents pour leurs encouragements attentionnés et conseils précieux durant toute ma scolarité. Merci également à mes frère et sœur, j'ai été très flatté qu'ils assistent à ma soutenance, et je suis ravi qu'elle leur ait plu.

Table des matières

Chapitre 1

Introduction

1.1 La robotique

On peut faire remonter la construction des premiers automates mécaniques à l'antiquité (Chapuis & Droz, 1949), que ce soit dans le but d'imiter des mouvements humains ou d'aider l'homme dans ses travaux pénibles. L'apparition des premiers ordinateurs au milieu du XX^{e} siècle et l'essor de l'intelligence artificielle ont permis de passer de ces automatismes mécaniques à des raisonnements informatiques de haut niveau capables d'adapter le comportement des machines à leur environnement. La robotique a alors été définie comme la science qui étudie « le lien intelligent entre la perception et l'action » (Siciliano & Khatib, 2008).

La *perception* est la construction de représentations utilisables du robot et de son environnement. Ce processus utilise les données fournies par des capteurs : caméras, radars, capteurs de force, microphones, etc. L'*action* désigne les évènements que le robot peut engendrer qui ont un impact sur son environnement : l'émission de lumières, de sons ou le mouvement *via* l'activation de moteurs. Nous appellerons *planification* les raisonnements qui permettent de passer d'une représentation du monde à une ou plusieurs actions. Cette thèse a pour objet la *planification de mouvement* pour des systèmes anthropomorphes : robots humanoïdes ou acteurs virtuels.

1.1.1 La robotique humanoïde

La robotique humanoïde a pour but de concevoir et contrôler des systèmes mécaniques directement inspirés par les capacités du corps humain. Ces robots ont généralement des chaînes articulaires semblables en forme et en structure à celle de l'homme, ainsi que des capacités de perception aussi similaires que possible aux capacités humaines. Soulignons les défis mécatroniques que posent la conception de tels systèmes, défis relevés en partie, et depuis peu grâce à la miniaturisation des moteurs et ordina-

teurs qui ont permis la construction de machines comme les robots HRP-2 (Kawada (Kaneko *et al.*, 2004)), Asimo (Honda (Sakagami *et al.*, 2002)) ou Partners (Toyota (Takagi, 2006)).

Les motivations pour la construction de ces systèmes sont variées. D'un point de vue industriel, un robot humanoïde, s'il est parfois moins efficace qu'un mécanisme spécialisé dans une certaine tâche, a l'avantage d'être plus généraliste, et bien adapté à la conduction de travaux dans des environnements conçus pour des humains. Par ailleurs, d'un point de vue social, les interactions homme-robot peuvent tirer parti des similarités physiques entre un humain et un robot humanoïde. Le robot peut communiquer par des gestes, des regards ou des attitudes imitant la communication non-parlée des humains entre eux. Ces attributs sont essentiels lorsqu'un robot doit évoluer dans un milieu public, ou quand il est utilisé à des fins de divertissement. Enfin, d'un point de vue scientifique, l'étude du mouvement et du corps humains par la robotique humanoïde est source de coopérations fructueuses avec les neurosciences, la psychologie du développement ou encore la linguistique.

1.1.2 L'animation graphique

Un autre champs d'application des méthodes présentées dans cette thèse est l'animation de personnages virtuels. Les applications principales de l'animation graphique sont le cinéma, les jeux vidéos et la simulation. La génération automatique de mouvement pour des acteurs virtuels a un but unique, qui prend des expressions variées : le réalisme. Dans le cas des industries de loisir, comme les films d'animation, un mouvement réaliste est tel qu'il semblera possible aux spectateurs. Dans le cas de la simulation, on exige du mouvement qu'il suive précisément les lois physiques, dans le but d'étudier le déroulement des évènements tels qu'ils se dérouleraient dans le monde réel. Parmi les techniques pour animer des personnages de façon réaliste, mentionnons la capture de mouvement, qui consiste à enregistrer dans un premier temps les données articulaires d'un humain qui exécute un mouvement pour le rejouer ensuite sur l'acteur virtuel. La qualité des résultats produits par la capture de mouvement en fait un outil de choix pour le cinéma ou les jeux vidéos.

1.1.3 La planification de mouvement

Au vu de la description donnée plus haut de ce qu'était la planification, on peut définir un algorithme de planification de mouvement par :

- ses entrées : les descriptions géométriques d'un système articulé et d'un environnement,
- sa sortie : une suite de mouvements qui résout une certaine tâche.

Malgré la différence de nature entre les robots humanoïdes et les acteurs virtuels, les problèmes que pose la planification de mouvement pour ces systèmes sont similaires. Le premier est la dimension des données traitées. Le corps humain possède en effet un grand

nombre d'articulations, et la génération de mouvement pour le corps complet nécessite d'explorer des espaces hautement dimensionnés, souvent trop complexes pour être représentés explicitement. En contrepartie, cette complexité articulaire rend ces systèmes hautement redondants, et leur permet de réaliser de nombreuses tâches. La deuxième difficulté inhérente aux systèmes anthropomorphes est la prise en compte de contraintes de stabilité. Pour garantir qu'une posture ou un mouvement est à l'équilibre statique ou dynamique, le planificateur de mouvement doit intégrer des outils sophistiqués de cinématique inverse.

1.2 Contributions

La première contribution de cette thèse est le développement et l'étude d'une méthode générique de planification de mouvement qui utilise des techniques statistiques de réduction de dimension. Cet algorithme a été conçu pour les systèmes hautement dimensionnés et les espaces contraints, et convient bien aux problèmes difficiles d'animation graphiques.

La deuxième contribution est le développement d'algorithmes de planification sous contraintes. Il s'agit d'une intégration d'outils de cinématique inverse hiérarchisée aux algorithmes de planification de mouvement randomisés. On illustre cette méthode sur des problèmes de manipulation par un robot humanoïde.

La troisième contribution est la généralisation des méthodes de planification sous contraintes à la locomotion.

Enfin, la dernière contribution présentée dans cette thèse est l'utilisation des méthodes précédentes pour résoudre des tâches de manipulation complexes par un robot humanoïde. Nous présentons en particulier un formalisme destiné à représenter les informations utilisables par un planificateur de mouvement . Nous avons présenté ce formalisme sous le nom d'« objets documentés ».

1.3 Plan de la thèse

Le chapitre 2 présente un état de l'art de la planification de mouvement randomisée. Il s'agit du type d'algorithmes sur lequel repose tout le travail présenté ici. Le chapitre 3 fait le point sur les méthodes de génération de mouvement spécifiques aux robots humanoïdes.

Le chapitre 4 présente notre contribution sur l'utilisation de la réduction de dimension en planification de mouvement randomisée. Le chapitre 5 détaille notre méthode pour planifier des mouvements sous contraintes de tâche, le chapitre 6 présente la généralisation de cette méthode à la planification de marche et enfin le chapitre 7 introduit et illustre le formalisme des objets documentés.

1.4 Publications associées à cette thèse

Article de revue :

– Sébastien Dalibard, Jean-Paul Laumond, **Linear Dimensionality Reduction in Random Motion Planning**, *International Journal of Robotics Research*, 2011

Conférences internationales avec comité de lecture :

– Sébastien Dalibard, Jean-Paul Laumond, **Control of probabilistic diffusion in motion planning**, *Algorithmic Foundation of Robotics VIII, Springer, 2009, p. 467–481*

– Sébastien Dalibard, Alireza Nakhaei, Florent Lamiraux, Jean-Paul Laumond, **Whole-body task planning for a humanoid robot : a way to integrate collision avoidance**, *9th IEEE-RAS International Conference on Humanoid Robots, 2009, p. 355–360*

– Sébastien Dalibard, Alireza Nakhaei, Florent Lamiraux, Jean-Paul Laumond, **Manipulation of Documented Objects by a Walking Humanoid Robot**, *10th IEEE-RAS International Conference on Humanoid Robots, 2010, p. 518 –523*

– Sébastien Dalibard, Antonio El Khoury, Florent Lamiraux, Michel Taïx, Jean-Paul Laumond, **Small-time controllability of a walking humanoid robot**, *11th IEEE-RAS International Conference on Humanoid Robots, 2011*

Chapitre 2

Méthodes d'échantillonnage en planification de mouvement

Ce chapitre présente le problème de la planification de mouvement sous sa forme la plus générale. Il décrit plus précisément la classe des algorithmes basés sur l'échantillonnage aléatoire. Cette présentation s'inspire de (LaValle, 2006). Des revues complètes des problèmes et méthodes en planification de mouvement peuvent aussi être trouvées dans (Choset *et al.*, 2005; Latombe, 1991).

2.1 Le problème du déménageur de piano

La robotique a pour but de concevoir des systèmes autonomes, capables de percevoir, raisonner, se déplacer et agir sur le monde qui les entoure. Un besoin fondamental pour un système robotique est la capacité à traduire des ordres donnés par un humain et exprimés à un haut niveau d'abstraction : « Va dans cette pièce, attrape cet objet », en une série de mouvements de bas niveau, qui décrivent de façon effective comment le robot va se mouvoir. La planification de mouvement, ou de trajectoire, s'efforce de répondre à ce besoin.

Le problème a été présenté dans la littérature sous le nom du *problème du déménageur de piano* dans (Schwartz *et al.*, 1987). Il peut s'écrire ainsi : étant donnés un environnement composé d'obstacles et un piano, est-il possible de déplacer le piano d'une position et orientation - appelées sa *configuration* q - à une autre, sans entrer en collision avec les obstacles. La figure 2.1 présente un acteur digital et un robot humanoïde tentant de répondre à cette question.

Figure 2.1 – Le robot humanoïde HRP-2 et un personnage virtuel déménageant un piano dans un environnement encombré.

2.2 L'espace des configurations

Au début des années 1980, (Lozano-Perez, 1983) a introduit le concept d'*espace des configurations* (noté $\mathcal{CS}$ dans la suite), qui est l'ensemble de toutes les configurations qu'un système peut adopter. Ce formalisme permet de traduire le problème du mouvement de corps dans le monde réel $\mathbb{R}^2$ ou $\mathbb{R}^3$ en celui d'un point dans un autre espace $\mathcal{CS} \subset \mathbb{R}^n$. La dimension de la variété $\mathcal{CS}$ est égale au nombre de variables indépendantes ou *degrés de liberté* dont la donnée à un instant t décrit entièrement la configuration du système considéré. Les obstacles dans le monde réel induisent des obstacles dans $\mathcal{CS}$. On définit $\mathcal{CS}_{\text{obstacle}} \subset \mathcal{CS}$ comme l'ensemble des configurations q telles qu'en q le robot est en collision avec un obstacle de son environnement. Le complémentaire de $\mathcal{CS}_{\text{obstacle}}$ est appelé l'espace libre, et noté $\mathcal{CS}_{\text{free}}$. Il s'agit de l'ensemble des configurations admissibles de $\mathcal{CS}$. Notons qu'on exclut de $\mathcal{CS}_{\text{free}}$ les configurations au contact des obstacles, ce qui implique que $\mathcal{CS}_{\text{free}}$ est un ouvert de $\mathcal{CS}$.

Muni de ce formalisme, on peut reformuler le problème de la planification de mouvement ainsi : étant donnés $\mathcal{CS}$, $\mathcal{CS}_{\text{obstacle}}$ et deux configurations q_{initial} et q_{final} appartenant

à $\mathcal{CS}_{\text{free}}$, existe-t-il un *chemin* reliant q_{initial} à q_{final}, c'est-à-dire une fonction continue $\tau : [0,1] \rightarrow \mathcal{CS}_{\text{free}}$ telle que $\tau(0) = q_{\text{initial}}$ et $\tau(1) = q_{\text{final}}$?

Plusieurs planificateurs ont été proposés pour répondre à cette question de manière exacte, en construisant des représentations explicites de $\mathcal{CS}_{\text{obstacle}}$. Des algorithmes de décomposition en cellules (Schwartz *et al.*, 1987), en diagramme de Voronoi ou graphes de visibilité (Latombe, 1991) ont été proposés. Le but est de construire un graphe dans $\mathcal{CS}_{\text{free}}$ (souvent appelé *roadmap* en anglais et noté $\mathcal{R}$ dans la suite), accessible depuis tout $q \in \mathcal{CS}_{\text{free}}$. Ainsi, la recherche d'un chemin entre q_{initial} et q_{final} est ramenée à la recherche d'un chemin dans un graphe. L'algorithme de Canny (Canny, 1988) construit $\mathcal{R}$ de façon plus efficace, sans passer par une décomposition en cellules de $\mathcal{CS}_{\text{free}}$, et résout le problème de la planification de mouvement en un temps simplement exponentiel en la dimension du problème. Une revue de ces planificateurs est disponible dans (Goodman & O'Rourke, 2004). Ces approches ont la propriété intéressante d'être exactes : après leur exécution, dont on peut donner un majorant en temps en fonction de la taille des données d'entrée, elles renvoient un chemin valide si et seulement s'il en existe un. On qualifiera cette propriété de *complétude*.

Le coût algorithmique de ces méthodes les rend inutilisables en pratique sur nombre de problèmes réels, où la complexité des modèles géométriques ou la dimension des systèmes considérés rend la taille d'une représentation explicite de $\mathcal{CS}_{\text{free}}$ trop grosse pour être calculée ou utilisée sur un ordinateur usuel.

Afin de dépasser les limites de ces méthodes exactes, des planificateurs randomisés ont été développés ces quinze dernières années. Bien qu'ils ne garantissent qu'une complétude plus faible, qualifiée de *probabiliste*, leur capacité à résoudre des problèmes complexes et hautement dimensionnés efficacement les a imposés en planification de mouvement. Le travail présenté ici s'inscrit pour l'essentiel dans le développement et l'amélioration de ces méthodes.

2.3 Planificateurs de mouvement par échantillonnage aléatoire

Le but de cette classe d'algorithmes est de construire un graphe aléatoire $\mathcal{R}$, dont les sommets sont des configurations libres de collision et dont les arêtes traduisent l'existence d'un chemin élémentaire (par exemple une ligne droite dans $\mathcal{CS}$) libre de collision entre deux configurations. On souhaite que $\mathcal{R}$ capture la topologie de $\mathcal{CS}_{\text{free}}$, c'est-à-dire qu'à chaque composante connexe de $\mathcal{CS}_{\text{free}}$ corresponde une et une seule composante connexe de $\mathcal{R}$. Le principe est de ne pas calculer de représentation explicite de $\mathcal{CS}_{\text{free}}$, mais seulement de valider ou d'invalider certaines configurations à l'aide d'un *détecteur de collisions*. On utilise ce détecteur comme une boîte noire afin que l'implémentation des algorithmes de planification eux-mêmes ne dépende pas des caractéristiques géométriques des systèmes pour lesquels on planifie.

Outre le détecteur de collision, on utilise une *méthode locale* qui prend en entrée

deux configurations, et génère un chemin élémentaire qui les relie. Cette méthode est liée au système considéré. Il peut s'agir d'une simple interpolation en ligne droite dans $\mathcal{CS}$, si le système peut se déplacer dans toutes les directions. Dans le cas de robots mobiles à roue par exemple, le robot ne peut pas se déplacer sur le côté sans manœuvrer. La méthode locale génère alors des trajectoires admissibles pour le système. La suite de la présentation sur l'échantillonnage fera l'hypothèse qu'on utilise une interpolation linéaire, la section 2.5 expliquera comment adapter ces algorithmes à d'autres méthodes locales.

2.3.1 Proabilistic Roadmaps (PRM)

La planification de mouvement par échantillonnage aléatoire été présentée pour la première fois dans (Kavraki *et al.*, 1996) sous le nom de *Probabilistic Roadmaps* (noté PRM dans la suite). L'algorithme PRM procède en deux phases :

1. Dans un premier temps, on génère des configurations aléatoires uniformément dans $\mathcal{CS}$, qu'on passe au détecteur de collisions. Si une configuration est valide, on l'ajoute à $\mathcal{R}$. On essaye ensuite de la relier par des lignes droites aux configurations déjà dans $\mathcal{R}$. Si un tel chemin est valide, on ajoute l'arête correspondante dans $\mathcal{R}$.
2. Afin d'échantillonner plus spécifiquement les passages contraints de $\mathcal{CS}_{\text{free}}$, on échantillonne ensuite des configurations aléatoires dans les voisinages des configurations de $\mathcal{R}$ présumées « difficiles ». La mesure de cette difficulté pour une configuration q est proportionnelle au nombre de fois qu'on a essayé de relier q à une autre configuration sans y parvenir.

La deuxième phase de cet algorithme, qui consiste à faire des hypothèses sur la frontière de la zone visible depuis $\mathcal{R}$ et à essayer d'étendre cette frontière en adaptant l'échantillonnage, est communément appelée dans la littérature une phase d'*expansion* de $\mathcal{R}$.

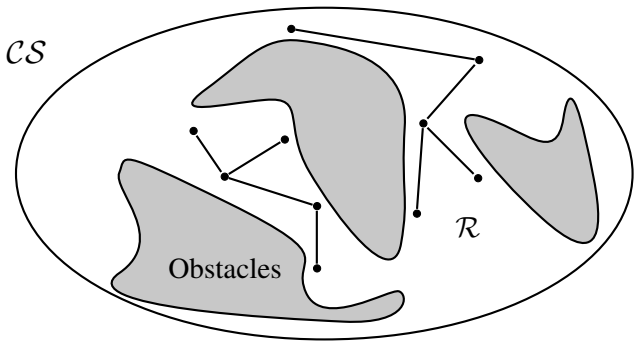


Figure 2.2 – Construction de $\mathcal{R}$ par l'algorithme PRM. Les obstacles sont représentés par les zones de couleur grise. Le graphe est composé de deux composantes connexes, qui devront être reliées pour capturer la topologie de $\mathcal{CS}_{\text{free}}$.

Le but de l'algorithme PRM est de construire $\mathcal{R}$ dans un premier temps pour pouvoir répondre à des requêtes multiples de planification de mouvement dans le même environnement dans un deuxième temps. La figure 2.2 présente un algorithme PRM en cours d'exécution. $\mathcal{CS}_{\text{obstacle}}$ est représenté par les zones grises. $\mathcal{CS}_{\text{free}}$ ne contient qu'une seule composante connexe, et $\mathcal{R}$ deux, donc l'algorithme n'est pas encore arrivé à une description satisfaisante de $\mathcal{CS}_{\text{free}}$.

2.3.2 Rapidly-Exploring Random Trees (RRT)

Suite au succès des PRM pour planifier des mouvements dans des espaces hautement dimensionnés, une classe d'algorithmes a donné lieu à d'importantes et fructueuses recherches : les stratégies d'expansion d'arbres aléatoires, présentées à la fois dans (Kuffner & LaValle, 2000) et (Hsu *et al.*, 1999), parmi lesquelles les *Rapidly-Exploring Random Trees* (notés RRT dans la suite). Dans ces algorithmes, $\mathcal{R}$ est un arbre, qu'on fait grossir dans $\mathcal{CS}_{\text{free}}$. À chaque itération de l'algorithme, on échantillonne une configuration aléatoire q_{rand} dans $\mathcal{CS}$, on cherche le nœud de $\mathcal{R}$ le plus proche de q_{rand} : q_{near} et on étend l'arbre aussi loin que possible depuis q_{near} dans la direction de q_{rand} tout en restant dans $\mathcal{CS}_{\text{free}}$. La figure 2.3 présente une étape d'extension de $\mathcal{R}$.

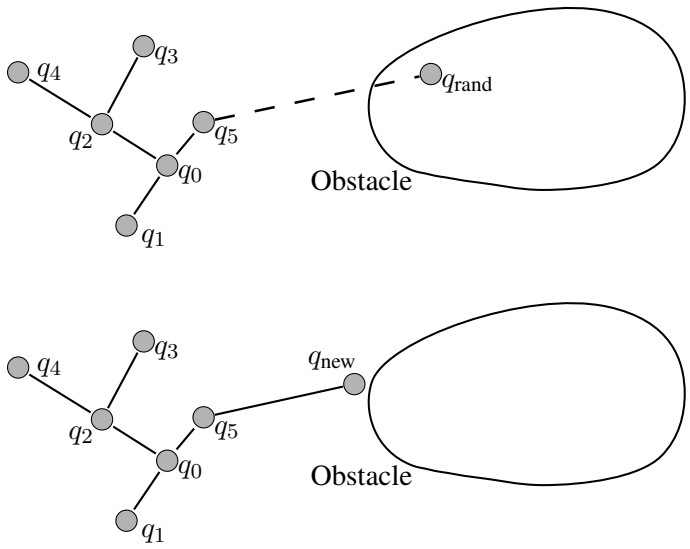


Figure 2.3 – Une étape d'extension d'un RRT.

L'algorithme 1 présente l'architecture d'un RRT. Une étape d'extension depuis q_{near} vers q_{rand} correspond à une expansion de $\mathcal{R}$ selon la terminologie utilisée pour les PRM. Nous utiliserons indifféremment l'un ou l'autre de ces termes dans la suite.

Parce qu'il n'utilise que des extensions aléatoires, au lieu d'un échantillonnage uniforme de $\mathcal{CS}$, l'algorithme RRT est mieux adapté que PRM à la planification dans certains espaces contraints, typiquement aux problèmes de désassemblage mécanique. Les chapitres suivant s'attarderont sur les différences entre ces deux algorithmes.

Algorithme 1 RRT(q_0)

$\mathcal{R}$.Init(q_0)
for i = 1 to K **do**
  $q_{rand} \leftarrow$ Rand(CS)
  $q_{near} \leftarrow$ Nearest($q_{rand}, \mathcal{R}$)
  $q_{new} \leftarrow$ Extend(q_{near}, q_{rand})
  $\mathcal{R}$.AddVertex(q_{new})
  $\mathcal{R}$.AddEdge(q_{near}, q_{new})
end for

L'algorithme présenté résout le problème de l'exploration de $\mathcal{CS}_{\text{free}}$. Toutefois, il ne répond pas précisément au problème de la planification de mouvement, c'est-à-dire à la recherche d'un chemin entre q_{initial} et q_{final}. La méthode la plus simple pour résoudre une requête de planification en utilisant l'algorithme RRT consiste à faire grossir un arbre enraciné en q_{initial} et à vérifier à chaque itération de l'algorithme si on peut relier q_{new} à q_{final} par un chemin élémentaire. Une approche souvent plus efficace consiste à faire grossir deux arbres dans $\mathcal{CS}_{\text{free}}$ enracinés respectivement en q_{initial} et q_{final}, et à vérifier régulièrement si on peut connecter ces deux arbres entre eux.

2.3.3 Optimisateurs de chemins aléatoires

Les chemins générés par ces algorithmes comportent beaucoup de « détours » à cause de la randomisation, et nécessitent généralement d'être optimisés avant de pouvoir être suivis par le système.

L'optimisation la plus courante repose sur une méthode dite *de raccourcis*, qui consiste à échantillonner des couples de points sur le chemin et à construire un chemin élémentaire les reliant. Si ce chemin élémentaire est plus court - ou moins coûteux, pour une fonction de coût associée au système - et sans collision, on remplace la partie correspondante du chemin de départ. La figure 2.4 illustre le déroulement d'une telle optimisation.

Cette méthode est assez sommaire, et facile à implémenter. Dans de nombreux cas, les résultats qu'elle donne sont suffisants. Les résultats expérimentaux présentés dans les chapitres suivants ont été générés avec cette technique. Récemment, des techniques plus sophistiquées d'optimisation ont été utilisées en planification de mouvement pour l'optimisation de trajectoires, par exemple dans (Ratliff *et al.*, 2009). Au prix de calculs plus compliqués - et souvent plus coûteux - ces méthodes peuvent fournir des solutions mieux optimisées. Un autre axe de recherche récent est l'adaptation des méthodes aléatoires pour garantir que les chemins générés sont quasi-optimaux. (Karaman & Frazzoli, 2010) présente l'algorithme RRT*, qui au prix d'une complexité asymptotiquement équivalente à celle du RRT - à une constante près - converge presque-sûrement vers un chemin optimal.

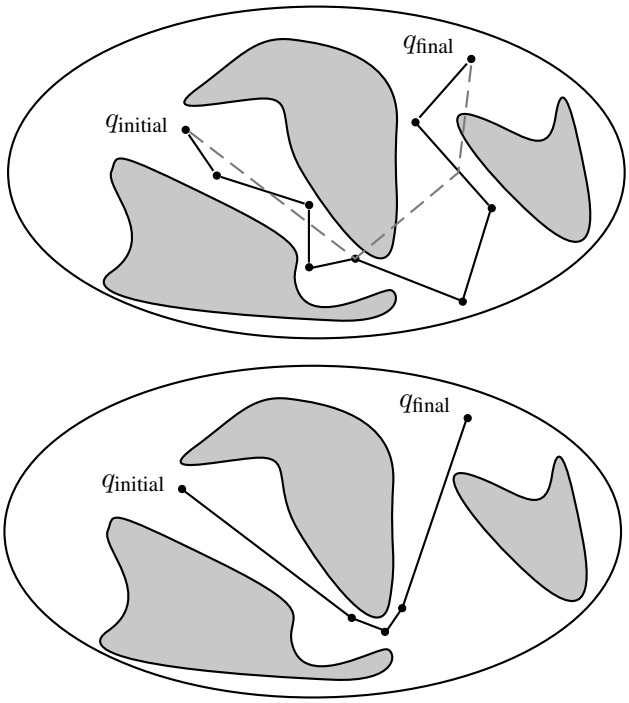


Figure 2.4 – Optimisation de chemin basée sur la méthode des raccourcis. Une fois qu'un chemin aléatoire reliant q_{initial} à q_{final} a été trouvé, on l'optimise en essayant des raccourcis le long du chemin. Sur la première figure, des pointillés gris représentent des raccourcis potentiels, dont il reste à vérifier qu'ils sont sans collision. Le premier raccourci (qui part de q_{initial}) est valide, mais les deux suivants intersectent les obstacles. La deuxième figure montre le résultat de plusieurs itérations de cette méthode.

2.4 Complexité des méthodes d'échantillonnage

Les méthodes présentées dans la section précédente ont la propriété d'être complètes en probabilité. Cela signifie que s'il existe une solution à un problème, la probabilité que l'algorithme la trouve en un temps inférieur ou égal à t tend vers 1 quand t tend vers $+\infty$. Cette propriété est plus faible que la complétude classique évoquée précédemment en ce que le fait que l'algorithme ne trouve pas de solution ne permet pas de décider si c'est parce qu'il n'en existe pas ou parce qu'on n'a pas attendu suffisamment longtemps.

La complétude probabiliste de PRM a été prouvée dans (Kavraki *et al.*, 1996) et celle de RRT dans (Kuffner & LaValle, 2000). L'idée dans les deux cas est que la distribution des configurations de $\mathcal{R}$ converge vers une distribution dense dans $\mathcal{CS}_{\text{free}}$. Ainsi, si un chemin existe dans $\mathcal{CS}_{\text{free}}$ reliant q_{initial} à q_{final}, après suffisamment d'itérations, le chemin sera approximé arbitrairement près par un chemin dans $\mathcal{R}$ et le problème sera résolu.

Peu de résultats existent dans la littérature sur la vitesse de convergence des algorithmes randomisés de planification de mouvement. La section qui suit présente la principale formalisation proposée pour modéliser le comportement des algorithmes basés

sur l'échantillonnage aléatoire.

2.4.1 Espaces expansifs

Nous reprenons ici une caractérisation de la difficulté à planifier un mouvement en fonction de paramètres de $\mathcal{CS}_{\text{free}}$. Ce formalisme a été présenté dans (Hsu *et al.*, 1999). Il s'appuie sur la notion de *visibilité*, critique en planification de mouvement. La région de visibilité d'une configuration $q \in \mathcal{CS}_{\text{free}}$ est l'ensemble des configurations q' de $\mathcal{CS}_{\text{free}}$ telles que le chemin élémentaire entre q et q' est sans collision. Pour une description détaillée de la notion de visibilité en planification de mouvement et sa relation avec la complexité de la planification de mouvement randomisée, on pourra se référer à (Siméon *et al.*, 2000).

La difficulté principale à laquelle un planificateur de mouvement peut être confronté est la présence de *passages étroits* dans $\mathcal{CS}_{\text{free}}$. En effet, lorsque le chemin à trouver passe par un tel passage, il est nécessaire d'échantillonner des configurations à l'intérieur du passage pour pouvoir résoudre le problème. Le volume d'un passage étroit étant petit devant celui de $\mathcal{CS}$, il faut parfois attendre longtemps avant d'échantillonner des configurations valides à l'intérieur du passage. La figure 2.5 présente un exemple d'espace de configurations qui comprend un passage étroit. Cet exemple et l'analyse de sa complexité ont été tirés de (Hsu, 2000). Le concept d'espace expansif formalise l'intuition qu'on vient de donner sur les difficultés que posent les passages étroits.

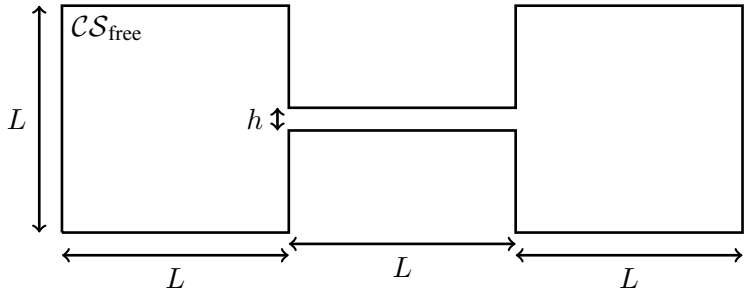


Figure 2.5 – Espace des configurations à deux dimensions. L'espace libre comprend un passage étroit ($h << L$). Planifier un chemin qui passe par le passage étroit est difficile car il faut générer des échantillons à l'intérieur du passage, qui est de faible volume comparé à $\mathcal{CS}$.

Notations : Pour tout sous-ensemble S de $\mathcal{CS}_{\text{free}}$, $\mu(S)$ est le volume de S. On normalise en supposant $\mu(\mathcal{CS}_{\text{free}}) = 1$. Par ailleurs, pour $q \in \mathcal{CS}_{\text{free}}$ on note $V(q)$ le sous-ensemble de $\mathcal{CS}_{\text{free}}$ visible depuis q.

Définition 1. *On dira qu'un espace libre $\mathcal{CS}_{free}$ est $(\alpha, \beta, \varepsilon)$-expansif si et seulement si, pour toute composante connexe F de $\mathcal{CS}_{free}$, on a :*

– $\forall q \in F$, $\mu(V(q)) > \varepsilon$

– *Pour tout S sous-ensemble connexe de F, l'ensemble*

$$LOOKOUT(S) = \{q \in S | \mu(V(q) \backslash S) \geq \beta \mu(F \backslash S)\}$$

est de volume μ(LOOKOUT(S))$\geq \alpha\mu(S)$.

La première condition traduit le fait que de toute configuration, on puisse voir au moins une fraction ε de l'espace libre.

En ce qui concerne la deuxième condition, $LOOKOUT(S)$ est le sous-ensemble de S dont on peut voir une fraction β du complémentaire de S. La condition est que ce sous-ensemble représente une fraction au moins α du volume de S. Les paramètres α et β expriment donc le volume relatif des ensembles de points qui peuvent contribuer à augmenter la taille des zones de visibilité lorsqu'on tire des configurations aléatoires à ajouter à $\mathcal{R}$.

Ainsi, si on prend pour S la région de visibilité de $\mathcal{R}$ à un instant donné, dans un espace $(\alpha, \beta, \varepsilon)$-expansif, avec α et β grands, on a beaucoup de chances d'échantillonner de nouvelles configurations qui augmenteront le volume de la région de visibilité de $\mathcal{R}$.

À l'inverse, des espaces comprenant des passages étroits ne pourront avoir que des α et β petits. Seule une faible fraction du volume de la zone qui est d'un côté du passage voit une fraction importante du complémentaire de cette zone à travers le passage.

(Hsu, 2000) énonce et démontre le résultat suivant :

Théorème 2.4.1. *On suppose que $\mathcal{CS}$ est $(\alpha, \beta, \varepsilon)$-expansif. Soit $\gamma \in]0, 1[$, soit S un ensemble de configurations, de cardinal $2n$ avec $n = \lceil 8 \ln(8/\varepsilon\alpha\gamma)/\varepsilon\alpha + 3/\beta + 2 \rceil$. Les configurations sont choisies indépendemment et uniformément dans $\mathcal{CS}_{free}$. On construit le graphe $\mathcal{R}$ de sommets ces configurations, en ajoutant des arêtes entre les configurations visibles l'une depuis l'autre. Alors avec une probabilité au moins $1 - \gamma$, la topologie de $\mathcal{R}$ capture celle de $\mathcal{CS}_{free}$, c'est-à-dire : tout sous-graphe de $\mathcal{R}$ constitué de configurations appartenant à la même composante connexe de $\mathcal{CS}_{free}$ est connexe.*

La propriété énoncée : « la topologie de $\mathcal{R}$ capture celle de $\mathcal{CS}_{\text{free}}$ » signifie que s'il existe une solution au problème de planification de mouvement considéré, on l'a trouvée. La dépendance de n en $\ln(\gamma)$ est intéressante, elle signifie que la probabilité de ne pas trouver de solution décroît exponentiellement avec le nombre d'itérations d'un algorithme PRM, par exemple. La dépendance en (α, β) est en $O(-\ln(\alpha)/\alpha + 1/\beta)$ qui tend vers $+\infty$ quand α ou β tendent vers 0. Les limites de ce résultat sont qu'il est difficile d'évaluer α, β et ε pour un $\mathcal{CS}$ donné, ce qui rend le théorème non effectif.

On peut toutefois donner l'ordre de grandeur de ces paramètres pour l'exemple simple de la figure 2.5. Ici, les points qui ont la plus faible visibilité sont à l'intérieur du passage étroit. Ils voient un volume $3Lh$, alors que le volume total de $\mathcal{CS}_{\text{free}}$ est en $\Theta(L^2)$, d'où $\varepsilon \sim (h/L)$. Si on considère le carré de gauche, seule une fraction de volume $\sim Lh$ voit des points extérieurs au carré, et ces points voient un volume $\sim Lh$ du complémentaire du carré. D'où $\alpha \sim h/L$ et $\beta \sim h/L$. Si on considère le même

exemple en dimension d, où le passage étroit est contraint dans k directions on trouve $\alpha, \beta, \varepsilon \sim (h/L)^k$.

Ce formalisme illustre à quel point les planificateurs de mouvement sont ralentis par les passage étroits et la dimension de $\mathcal{CS}$.

2.5 Commandabilité en temps petit

Comme mentionné précédemment, certains systèmes robotiques utilisent des méthodes locales différentes d'une interpolation en ligne droite. Un système est dit *commandable* si pour toute paire de configuration (q_1, q_2), il peut atteindre q_2 en partant de q_1, c'est-à-dire si la méthode locale appelée sur (q_1, q_2) retourne un chemin. Il est dit *commandable en temps petit* si pour tout temps $T > 0$ fixé, pour tout q, l'ensemble des configurations accessibles depuis q en un temps inférieur à T forme un voisinage de q.

En termes géométriques, et en supposant de manière naturelle que la vitesse et l'accélération du système sont bornées, cela signifie que pour toute configuration q donnée, pour tout voisinage V de q, l'ensemble des configurations q' telles que le chemin élémentaire entre q et q' est inclus dans V forme un voisinage v de q. La figure 2.6 illustre cette propriété.

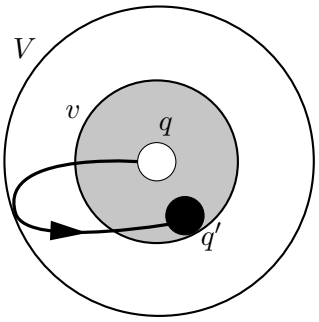


Figure 2.6 – Commandabilité en temps petit d'un système. L'ensemble des configurations q' accessibles depuis q par des chemins élémentaires inclus dans V forme un voisinage v de q.

La propriété de commandabilité en temps petit d'un système garantit qu'un chemin τ quelconque inclus dans $\mathcal{CS}_{\text{free}}$ peut être approximé par le système en restant dans $\mathcal{CS}_{\text{free}}$. En effet, pour tout point q de τ, il existe un ouvert $V \subset \mathcal{CS}_{\text{free}}$ contenant q (on rappelle que $\mathcal{CS}_{\text{free}}$ est ouvert). À V correspond un voisinage v de configurations accessibles depuis q en restant dans V par la méthode locale du système. L'ensemble de ces v forme un recouvrement de τ qui est compact, on peut donc en extraire un sous-recouvrement fini auquel correspond une suite finie de manœuvres qui approximent donc le chemin sans entrer en collision avec les obstacles.

La conséquence de cette propriété est qu'on peut raisonner en deux temps pour les systèmes dont la méthode locale n'est pas linéaire :

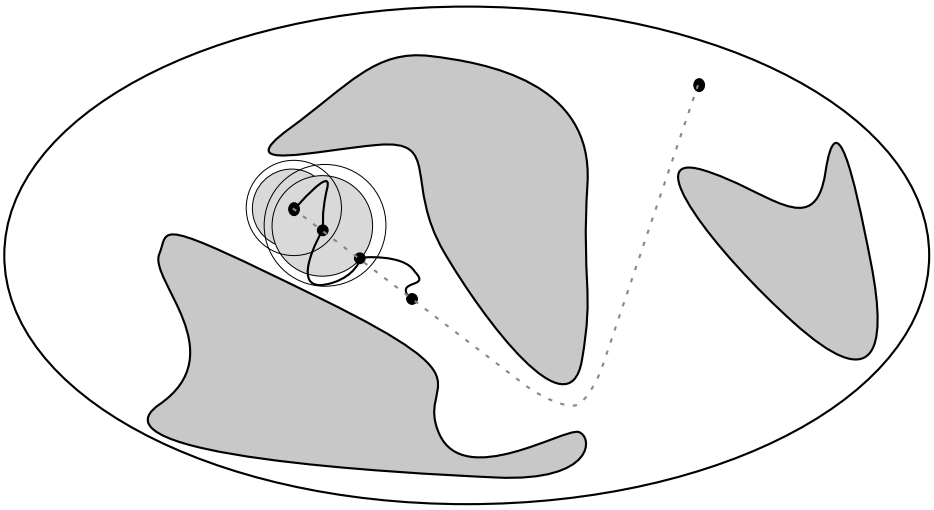


Figure 2.7 – Application de la commandabilité en temps petit d'un système. Le chemin initial (en pointillés gris) est approximé par une trajectoire admissible par le système (en trait plein noir) sans entrer en collision avec les obstacles.

1. on résout le problème géométrique en utilisant les algorithmes d'échantillonnage et une méthode locale linéaire,

2. on suit le chemin renvoyé par (1) en restant dans $\mathcal{CS}_{\text{free}}$ par une succession de manœuvres admissibles par le système.

Remarque : Ces raisonnements ne sont pas nécessairement valides au voisinage de points singuliers du système, des adaptations ont été présentées dans la littérature pour adapter les lois de contrôle aux systèmes singuliers, citons par exemple (Sekhavat & Laumond, 1998; Vendittelli *et al.*, 2004).

La figure 2.7 illustre cette stratégie en deux temps. Le chemin trouvé par le planificateur est représenté par des pointillés gris. Il est approximé par une trajectoire admissible par le système (en trait plein noir) en allant itérativement le plus loin possible le long du chemin initial tout en garantissant l'absence de collision avec les obstacles.

2.6 Limites et adaptations des algorithmes d'échantillonnage

Le formalisme des espaces expansifs présenté dans ce chapitre valide l'intuition selon laquelle les passages étroits sont une source de difficulté pour les planificateurs par échantillonnage. Toutefois, il ne permet pas de détecter la présence de passage dans $\mathcal{CS}$ et donc d'adapter la recherche aléatoire. Le chapitre 4 présente une contribution à ce sujet.

2.6.1 Planification pour chaînes fermées et sous contraintes

Une des principales limites des méthodes d'échantillonnages telles qu'on les a présentées dans ce chapitre est la prise en compte de contraintes inhérentes au système. Parfois, l'ensemble des configurations valides d'un système forment un sous-ensemble de $\mathcal{CS}$ de volume nul. Les échantillons aléatoires ont alors une probabilité 0 d'être valides. Ce problème a été présenté sous le nom d'*échantillonnage pour chaînes fermées*. Une des solutions consiste à paramétrer la sous-variété correspondant aux configurations vérifiant les contraintes, puis à échantillonner l'espace des paramètres. Parfois, cette sous-variété ne peut pas être paramétrée, ce qui a conduit à l'utilisation de méthodes randomisées pour la génération d'échantillons qui vérifient les contraintes, voir par exemple (Cortes *et al.*, 2002; Han & Amato, 2000; Yakey *et al.*, 2001).

Plus récemment, un problème similaire a été présenté sous le nom de *planification sous contraintes de tâches*. Ce problème apparaît quand des tâches doivent être réalisées par le robot au cours de la planification, par exemple lors de l'ouverture d'une porte, ou d'un suivi de trajectoire. La réalisation de ces tâches impose - de même que pour l'échantillonnage pour les chaînes fermées - que les configurations considérées par les planificateurs de mouvement appartiennent à des sous-variétés de $\mathcal{CS}$ de volume nul. (Simeon *et al.*, 2004; Stilman, 2007; Oriolo & Vendittelli, 2009) présentent des adaptations des planificateurs de mouvement randomisés, adaptés à la planification sous contraintes de tâches de manipulation. Le chapitre 5 présente une contribution à ce domaine de recherche, orientée plus spécifiquement vers la robotique humanoïde.

Chapitre 3

Génération de mouvement pour systèmes anthropomorphes

Les robots humanoïdes, ainsi que les acteurs digitaux, sont des systèmes complexes et hautement redondants. Leur grand nombre de degrés de liberté leur permet de se déplacer, en marchant et de résoudre de nombreuses tâches de manipulation : attraper et déplacer des objets, ouvrir des portes ou des tiroirs, *etc*.

L'utilisation de ces vastes capacités, qu'on souhaiterait équivalentes aux capacités physiques du corps humain, sont un défi posé aux planificateurs de mouvement. Comme nous l'avons évoqué précédemment, la taille des structures utilisées par un planificateur suit de façon exponentielle le nombre de degrés de liberté du système pour lequel on planifie. S'ajoutent à ce coût certaines contraintes spécifiques aux systèmes anthropomorphes. La formulation des problèmes de planification, bien souvent, ne suit pas celle du problème canonique du déménageur de piano. On n'attend pas d'un robot humanoïde qu'il atteigne une certaine configuration, mais plutôt qu'il résolve une tâche, exprimée dans son espace de travail (et pas dans $\mathcal{CS}$). Par ailleurs, la station debout comme la locomotion posent des problèmes d'équilibre - statique ou dynamique - qui ne peuvent généralement pas être résolus par un échantillonnage naïf de $\mathcal{CS}$.

De nombreux outils et formalismes spécifiques ont été développés pour résoudre les problèmes propres aux systèmes anthropomorphes. Ce chapitre fait le point sur ces outils. Nous présentons dans un premier temps le formalisme de la cinématique inverse hiérarchisée, qui répond au besoin de générer un mouvement résolvant une tâche plutôt qu'atteignant une configuration donnée. Ces algorithmes permettent aussi de générer des postures respectant des contraintes d'équilibre. Dans une deuxième section, nous présentons un modèle utilisé pour générer des mouvements de marche à l'équilibre dynamique. La troisième section présente les possibilités de couplage de la locomotion et de la manipulation. Tous ces algorithmes ne permettent pas en tant que tel de résoudre

le problème de l'évitement d'obstacles, nous parlerons ici de génération de mouvement plutôt que de planification. Une dernière section fait le point sur les adaptations de ces méthodes à la planification et à l'évitement d'obstacles.

3.1 Résolution de tâches de cinématique inverse hiérarchisées

Nous présentons ici rapidement le formalisme classique de la cinématique inverse avec priorités. La description que nous en faisons ainsi que les algorithmes que nous avons utilisés sont présentés plus en détail dans (Kanoun, 2009).

3.1.1 Cinématique inverse

La cinématique est l'étude du mouvement des corps sans considérer leur inertie ni les forces agissant sur eux. Étant donné un système cinématique composé de plusieurs segments reliés entre eux par des articulations, nous appelerons cinématique directe le calcul d'une propriété d'un de ces segments à partir des données articulaires, par exemple sa position. Nous parlerons d'*espace des configurations*, noté $\mathcal{CS}$ pour l'ensemble des données articulaires et d'*espace de la tâche*, ou *espace de travail*, noté $\mathcal{WS}$, pour l'ensemble dans lequel on exprime le résultat de la cinématique directe, par exemple, pour la position d'un point en 3 dimensions : $\mathbb{R}^3$.

Le problème de la cinématique inverse est de trouver une configuration articulaire satisfaisant une propriété dans l'espace de travail.

Considérons une structure cinématique à n degrés de libertés, décrite par sa configuration $q = (q_1, \ldots, q_n) \in \mathcal{CS}$. Nous représentons une tâche de cinématique inverse par une fonction différentiable : $T : \mathcal{CS} \rightarrow \mathcal{WS}$, avec $\mathcal{WS} = \mathbb{R}^m$ pour laquelle on souhaite trouver q telle que $T(q) = 0$. On dira que m est la dimension de la tâche. Les tâches qu'on souhaite exécuter en robotique s'expriment souvent par des fonctions non-linéaires, dont on ne connaît pas les fonctions réciproques. Afin de résoudre de tels problèmes, on est amené à utiliser des méthodes numériques.

Supposons que le système se trouve en une configuration q_0 telle que $T(q_0) = c \neq 0$, en évaluant la jacobienne de T : $J = \dfrac{\partial T}{\partial q}(q)$, nous pouvons calculer des petites variations δq qui tendent à résoudre la tâche. Nous choisissons pour δq la solution du système linéaire :

$$J\delta q = -\lambda T(q) \tag{3.1}$$

où $\lambda \in \mathbb{R}_+^*$.

Dans la suite, nous noterons $\delta T = -\lambda T(q)$: il s'agit de la variation de l'erreur qu'on souhaite atteindre en appliquant δq.

Quand $(n = m)$ et $(\lambda = 1)$, l'algorithme qui consiste à itérer 3.1 pour trouver une approximation d'une racine de T est appelé la méthode de Newton. Dans le cas plus général présenté ici, nous parlerons de descente de gradient.

La figure 3.1 illustre la recherche d'une racine de $T : \mathbb{R} \rightarrow \mathbb{R}$.

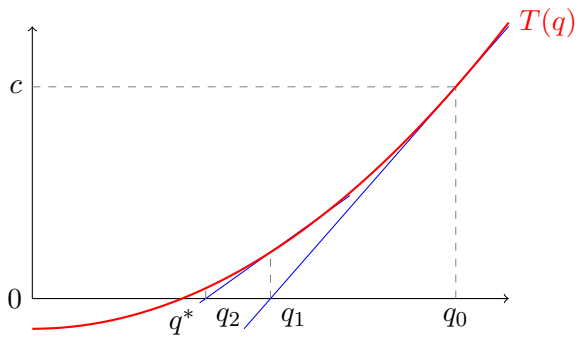


Figure 3.1 – Recherche d'une racine q^* de T par la méthode de Newton.

Si J est de rang plein et que $m \leq n$, il existe des solutions à 3.1. Sous certaines conditions sur le point de départ q_0 et sur la régularité de T, la descente de gradient converge vers un q^* satisfaisant $T(q^*) = 0$. Dans le cas de la méthode de Newton, lorsque la suite obtenue converge, la convergence est quadratique.

Si $m > n$, il peut ne pas exister de solution à 3.1. Si en une configuration q, J est singulière, nous dirons abusivement que q est singulière. La descente de gradient est alors numériquement instable. Des méthodes ont été présentées dans la littérature pour résoudre les problèmes d'instabilités numériques, voir par exemple (Nakamura & Hanafusa, 1986).

3.1.2 Tâches hiérarchisées

Considérons une tâche T_1, de dimension $m_1 < n$ et de matrice jacobienne J_1 et une deuxième tâche T_2 de dimension m_2 telle que $(m_1 + m_2 \leq n)$ et de matrice jacobienne J_2. On souhaite réaliser T_1 et T_2, avec T_1 prioritaire par rapport à T_2. Les solutions de 3.1 pour T_1 forment un espace affine de dimension $(n - m_1)$. Notons $J_1^{\#}$ la pseudo-inverse de J_1 : $J_1^{\#} = J_1^T(J_1 J_1^T)^{-1}$. Cet espace affine S_1 a pour équation :

$$S_1 = \left\{ \delta q \in \mathbb{R}^n | \exists z \in \mathbb{R}^n \text{ t.q. } \delta q = J_1^{\#} \delta T_1 + (I - J_1^{\#} J_1) z \right\} \tag{3.2}$$

$J_1^{\#}\delta T1$ est la solution de moindre norme de 3.1 et $(I - J_1 J_1^{\#})$ est le projecteur sur l'espace linéaire qui porte S_1. On peut maintenant exprimer le fait que T_1 doit être réalisée avec une priorité supérieure à T_2 en cherchant une solution à 3.1 pour T_2 qui appartienne à S_1. En réinjectant la forme des solutions de 3.2 dans 3.1 qu'on résout pour T_2, on obtient :

$$\delta T_2 - J_2 J_1^{\#} \delta T_1 = J_2 (I - J_1^{\#} J_1) z \tag{3.3}$$

qu'il faut résoudre en z.

Notons $N_1 = (I - J_1^{\#} J_1)$ et $\delta q_1 = J_1^{\#} \delta T1$, d'après (Nakamura & Hanafusa, 1986) la solution de moindre norme de 3.3 s'écrit :

$$\delta q = \underbrace{J_1^{\#} \delta T1}_{\delta q_1} + \overbrace{(J_2 N_1)^{\#} (\delta T_2 - J_2 \delta q_1)}^{\delta q_2} \tag{3.4}$$

Le deuxième terme (δq_2) correspond à un déplacement pour résoudre T_2 dans l'espace nul de $J_1^{\#}$, donc qui ne contrarie pas la première tâche.

On peut généraliser ce processus à k tâches $(T_1, \ldots, T_k)$. L' algorithme 2 présente les étapes de ce calcul.

Algorithme 2 Solve k prioritized linear equality systems

$n \leftarrow$ dimension of the kinematic structure
$N_0 \leftarrow I_n$ ($n \times n$ identity matrix)
$\delta q \leftarrow 0$
for $i = 1$ to k **do**
  Compute $(J_i, \delta T_i)$
  $\hat{J}_i \leftarrow J_i N_{i-1}$
  $N_i \leftarrow N_{(i-1)} - \hat{J}_i^{\#} \hat{J}_i$
  $\delta q_i \leftarrow \hat{J}_i^{\#} (\delta T_i - J_i \delta q_{i-1})$
  $\delta q \leftarrow \delta q + \delta q_i$
end for

La figure 3.2 présente un exemple de résultat d'un algorithme de cinématique inverse. Les contraintes à respecter sont l'équilibre statique, et la tâche consiste à amener la main droite à la position indiquée par le cube rouge.

Figure 3.2 – Réalisation d'une tâche d'atteinte pour la main droite en conservant l'équilibre statique, c'est-à-dire, les deux pieds au sol et le centre de masse projeté au centre de son polygone de sustentation.

3.2 Marche et équilibre dynamique d'un robot humanoïde

La section précédente a présenté un formalisme destiné à tirer parti de la redondance des systèmes anthropomorphes. Une autre caractéristique importante des robots humanoïdes est leur sous-actionnement. En effet, le contrôle des degrés de liberté non-articulaires, c'est-à-dire, des six degrés de liberté qui définissent la position et l'orientation du robot dans le repère du monde, ne peut se faire qu'indirectement, en marchant. La marche pose le problème de l'équilibre, statique ou dynamique. Cette section présente un critère géométrique couramment utilisée pour étudier l'équilibre d'un robot : le *Zero-Moment Point*. Nous présenterons ensuite une méthode de génération de trajectoires de marche à l'équilibre dynamique.

3.2.1 Zero-Moment Point (ZMP)

Considérons un système articulé, composé de K corps rigides $(C_k)_k$. On se place dans le repère du monde. Pour chaque corps C_k, notons m_k sa masse, x_k la position de son centre de masse, $\mathbb{I}_k$ sa matrice d'inertie, R_k sa matrice de rotation et ω_k son vecteur vitesse de rotation instantanée. Notons aussi m la masse totale du système et x_G la position de son centre de masse.

On peut écrire le torseur dynamique du système :

$$\begin{pmatrix} T \\ R \end{pmatrix} = \begin{pmatrix} \sum m_k \ddot{x}_k \\ \sum x_k \wedge m_k \ddot{x}_k + R_k \mathbb{I}_k \dot{\omega}_k - R_k \left((\mathbb{I}_k \omega_k) \wedge \omega_k \right) \end{pmatrix}$$

Dans le cadre de la marche, les forces appliquées au système sont son poids et les réactions du sol aux points de contact. Notons $(p_i)_i$ l'ensemble des points de contact, $(f_i)_i$ les forces de contact correspondantes et g l'accélération de la pesanteur. Le torseur d'action de ces forces sur le système s'écrit :

$$\begin{pmatrix} \sum m_k g + \sum f_i \\ \sum m_k x_k \wedge g + \sum p_i \wedge f_i \end{pmatrix}$$

Le principe fondamental de la dynamique stipule que le torseur dynamique du système est égal au torseur d'action des forces appliquées au système. Nous nous intéressons à des mouvements de marche sur sol plan et horizontal (dont on note la normale unitaire n) où le robot n'est en contact avec le sol qu'avec ses pieds. Les mouvements ne contiennent pas de sauts et sont sans glissements. Les forces de contact du sol sur les pieds sont dirigées vers le haut. Nous qualifierons ces conditions de *conditions de marche*. Sous ces contraintes, l'analyse présentée dans (Wieber, 2002) s'applique et garantit qu'un mouvement de marche est à l'équilibre si et seulement si le point défini par :

$$\frac{mgx_G + n \wedge R}{mg + T.n}$$

se projette verticalement sur le sol à l'intérieur de l'enveloppe convexe des points de contact (appelée dans la suite le *polygone support*).

Lorsque cette condition est vérifiée, la projection verticale de ce point sur le sol est appelée en robotique le *Zero Moment Point* (noté ZMP dans la suite). Cette notion a été présentée dans (Vukobratović *et al.*, 1990).

Notons que dans le cas d'une posture statique, c'est-à-dire quand le torseur dynamique est nul, cette condition se réduit au fait que le centre de masse se projette verticalement à l'intérieur du polygone support.

3.2.2 Modèle du « chariot sur une table »

Le ZMP dans sa formulation exacte est difficile à contrôler. Nous présentons ici un modèle approximatif de robot humanoïde qui conduit à un contrôle simple du ZMP. Il a été présenté dans (Kajita *et al.*, 2003) sous le nom de « chariot sur une table ». La structure cinématique du robot humanoïde est approximée par un chariot ponctuel, de masse celle du robot, se déplaçant sur une table de masse nulle à une hauteur constante z_c. Le pied de la table a la forme du polygone support du robot. Notons (x, y, z_c) les coordonnées du chariot. L'analyse présentée dans (Kajita *et al.*, 2003) donne les coordonnées (p_x, p_y) du ZMP en fonction de (x, y) (g est ici la norme de l'accélération de la pesanteur) :

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x - \frac{z_c}{g}\ddot{x} \\ y - \frac{z_c}{g}\ddot{y} \end{pmatrix} \tag{3.5}$$

La figure 3.3 illustre ce modèle simplifié. Sur cet exemple, le système n'est pas à l'équilibre statique : le centre de masse du robot n'est pas à la verticale de son polygone support. Toutefois, une accélération horizontale du centre de masse fait que le ZMP est bien défini à l'intérieur du polygone support, ce qui garantit un équilibre dynamique.

L'expression simple de l'équation 3.5 permet de contrôler le ZMP via la position du centre de masse du robot. (Kajita *et al.*, 2003) propose une méthode basée sur la commande prédictive. Muni d'un contrôleur de ZMP, on peut planifier une trajectoire de marche à l'équilibre dynamique comme suit :

1. On planifie des empreintes de pas, paramétrées en temps.
2. On construit une trajectoire de ZMP en fonction de ces empreintes pour garantir l'équilibre dynamique. Pendant les phases de simple support, on fixe le ZMP au centre du pied de support, pendant les phases de double support, on déplace le ZMP d'un pied à l'autre.
3. On passe cette trajectoire de ZMP au contrôleur qui génère une trajectoire articulaire corps-complet réalisant le mouvement de marche souhaité.

La figure 3.4 présente une suite d'empreintes de pas et la trajectoire planifiée du ZMP correspondante.

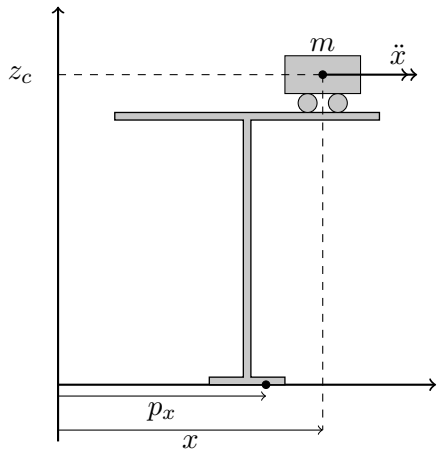


Figure 3.3 – Modèle simplifié du chariot sur une table. Le chariot représente le centre de masse du robot, qui peut se déplacer librement dans un plan horizontal. Le pied de la table représente le contact du robot avec le sol. Le système n'est pas à l'équilibre statique, mais la condition d'équilibre dynamique du ZMP est vérifiée.

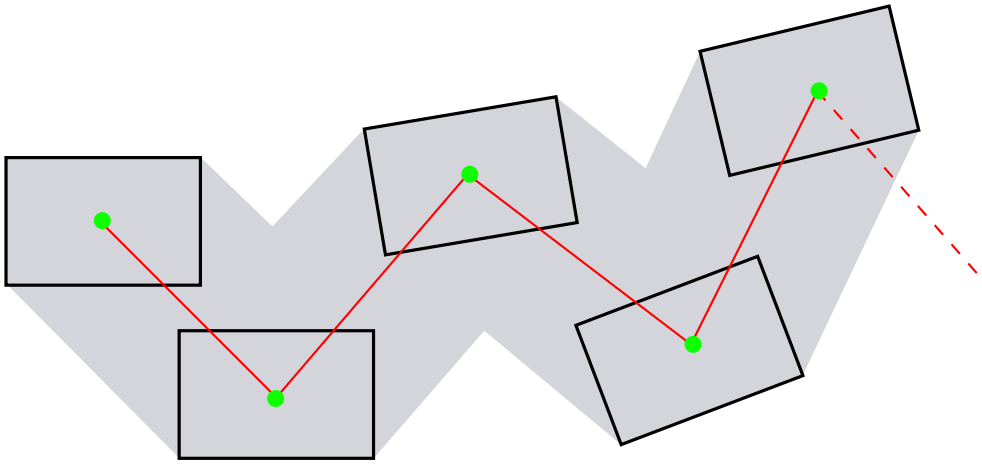


Figure 3.4 – Planification du ZMP pour qu'il reste à l'intérieur du polygone support. Les disques verts représentent une position fixe pendant une phase de simple support, les lignes rouges représentent le chemin parcouru par le ZMP d'un pied à l'autre pendant les phases de double support.

3.3 Marche et manipulation

Les méthodes des deux sections précédentes peuvent être couplées pour générer des mouvements de marche au cours desquels le robot accomplit des tâches de manipulation. Cette section couvre deux types d'approches qu'on trouve dans la littérature. La première est plus heuristique, et suppose que l'utilisateur enrichisse le modèle du robot avec de l'information sur les degrés de liberté dédiés à la marche ou à la manipulation,

respectivement. La seconde est générale.

3.3.1 Décomposition fonctionnelle

(Esteves Jaramillo, 2007) propose de résoudre les problèmes de manipulation pendant la marche au moyen d'une *décomposition fonctionnelle* des degrés de liberté du système anthropomorphe. Les degrés de liberté des jambes sont dédiés à la marche, ceux des bras à la manipulation et ceux du torse à l'adaptabilité. La figure 3.5, tirée de (Esteves Jaramillo, 2007) illustre ce principe.

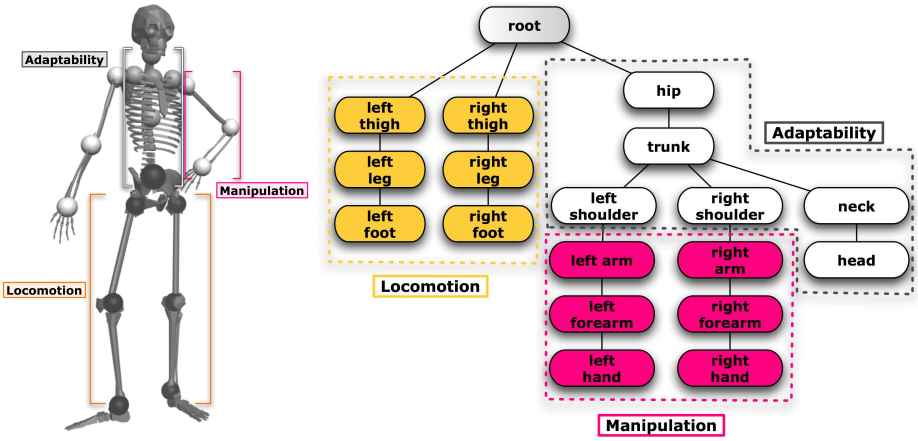


Figure 3.5 – Décomposition fonctionnelle d'un système anthropomorphe. L'ensemble des degrés de liberté du système est décomposé en trois groupes : locomotion, manipulation et adaptabilité. Figures tirées de (Esteves Jaramillo, 2007).

Munie de cette représentation, la planification commence par résoudre le problème de manipulation avec le haut du corps, puis anime les degrés de liberté liés à la locomotion pour suivre le mouvement.

L'intérêt d'une telle représentation est qu'elle rend la planification de mouvement rapide en réduisant sa complexité algorithmique. Cette simplification a un prix puisqu'elle ne permet pas de résoudre les problèmes de manipulation où l'ensemble des degrés de liberté sont nécessaires à la réalisation de la tâche. Ainsi, si le personnage doit attraper un objet sur le sol, la distinction entre les degrés de liberté utiles à la locomotion et ceux utiles à la manipulation n'est pas pertinente, car les degrés de liberté des jambes servent à atteindre l'objet avec la main.

3.3.2 Cinématique inverse généralisée à la marche

(Yoshida *et al.*, 2006) présente une combinaison des algorithmes de cinématique inverse hiérarchisée et du contrôle du ZMP présenté dans (Kajita *et al.*, 2003). Lors de la génération d'un mouvement de manipulation qui contient des pas, les tâches de plus haute priorité sont celles qui concernent l'équilibre dynamique :

– position du centre de masse,
– position et orientation du pied en vol.

Viennent ensuite les tâches concernant la manipulation.

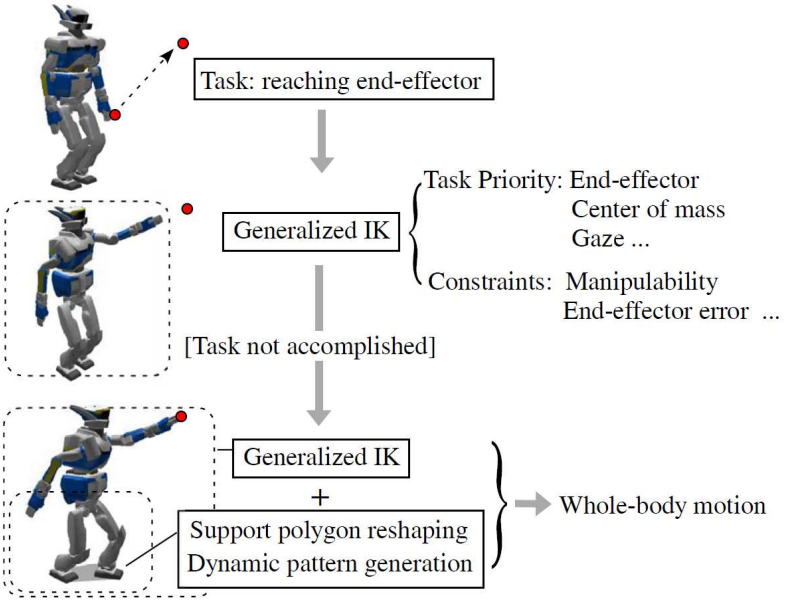


Figure 3.6 – Cinématique inverse hiérarchisée généralisée à la marche. La réalisation de la tâche d'atteinte nécessite que le robot fasse un pas. La position de ce pas est décidée de façon heuristique. Ensuite un même algorithme de résolution de tâches de cinématique inverse génère le mouvement d'atteinte et garantit l'équilibre dynamique. Figure tirée de (Yoshida *et al.*, 2006).

La figure 3.6, tirée de (Yoshida *et al.*, 2006) illustre un schéma global de génération de mouvement de manipulation qui nécessite un pas. La génération de mouvement comprend :

– une heuristique pour déterminer la position du pas nécessaire à la réalisation de la tâche,
– la résolution de tâche de cinématique inverse hiérarchisée, qui inclut un contrôleur de ZMP

Dans (Kanoun *et al.*, 2010), une extension de ce travail propose un placement automatique des empreintes de pas, en intégrant au calcul de cinématique inverse une chaîne articulaire virtuelle composée des emplacements des empreintes de pas. La figure 3.7 présente un exemple de posture d'atteinte calculée conjointement avec les empreintes de pas qui y mènent.

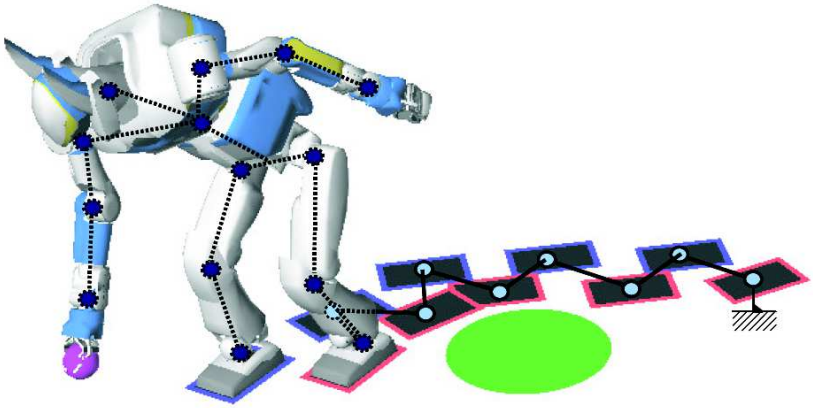


Figure 3.7 – Calcul automatique des empreintes de pas nécessaires à la réalisation d'une tâche d'atteinte. Le calcul se fait en intégrant la position des pas comme une chaîne articulaire virtuelle dans le calcul de cinématique inverse. Figure tirée de (Kanoun *et al.*, 2010).

3.4 Prise en compte de l'évitement d'obstacles

Les méthodes qu'on vient de présenter n'ont pas été conçues pour planifier des mouvements évitant des obstacles. Nous présentons dans cette section quelques adaptations de ces techniques pour résoudre le problème de planification de mouvement pour un robot humanoïde.

3.4.1 Navigation

La planification d'un mouvement de marche pour un système anthropomorphe dans un environnement comprenant des obstacles est un problème intrinsèquement difficile, en raison des nombreux degrés de liberté du système et de sa complexité géométrique.

Une solution qu'on trouve souvent dans la littérature consiste à ne considérer qu'une version simplifiée du problème : une boîte englobant le robot se déplaçant dans le plan. Cette simplification ramène le problème de la navigation d'un robot humanoïde à celui de la planification pour un système à trois degrés de liberté (x, y, θ).

Une fois qu'on a trouvé un chemin sans collisions pour la boîte englobante, on génère une animation de marche grâce aux techniques évoquées précédemment. Cette méthode est utilisée dans (Yoshida *et al.*, 2007) pour la navigation du robot humanoïde HRP-2 ou encore dans (Pettré *et al.*, 2003) pour planifier la locomotion d'un acteur digital. (Kanehiro *et al.*, 2004) propose une extension de ces méthodes dans laquelle la forme de la boîte englobante peut changer pour traduire les attitudes possibles du robot : marche debout, de côté, accroupie, *etc*.

3.4.2 Planification de pas

Une autre simplification possible pour planifier des mouvements de marche consiste à ne considérer comme degrés de liberté que les positions des empreintes de pas. Le détecteur de collision valide les emplacements des pas et ne concerne pas le mouvement corps complet du robot. Cette stratégie a été présentée dans (Chestnutt *et al.*, 2005), ici, les empreintes de pas sont planifiées à l'aide d'un algorithme A*.

3.4.3 Résolution de contraintes unilatérales par cinématique inverse

Plusieurs travaux ont intégré les contraintes d'évitement d'obstacles au formalisme de la cinématique inverse. Il existe essentiellement deux façons de le faire :

- soit sous forme de champs de potentiel (Khatib *et al.*, 2004), c'est-à-dire une fonction de q qui tend vers $+\infty$ quand q est presque en collision
- soit en généralisant le formalisme présenté en section 3.1 à des tâches définies par des inégalités (Kanoun *et al.*, 2009).

Notons que sur la figure 3.7, des tâches d'inégalités ont été ajoutées pour que les empreintes de pas évitent un disque vert sur le sol. Le chapitre 5 reviendra sur cette seconde approche, en la comparant à une utilisation simultanée des méthodes de cinématique inverse et de l'échantillonnage aléatoire.

3.4.4 Planification sous contraintes

Les travaux mentionnés dans la section 2.6.1 ont récemment été adaptés à la planification de mouvement pour robots humanoïdes. (Berenson *et al.*, 2011) présente un formalisme et des algorithmes pour planifier des mouvements pour humanoïdes qui respectent des contraintes de tâches. Le travail présenté en chapitre 5 s'inscrit précisément dans ce cadre.

3.4.5 Planification au contact

Il convient enfin, paradoxalement dans cette section, de mentionner des travaux qui planifient des mouvements en collision. Dans certains environnements, un robot humanoïde peut tirer parti des obstacles en s'appuyant dessus. Pour attraper un objet sur une table, par exemple, poser une main sur la table peut étendre l'espace accessible par l'autre main. (Escande *et al.*, 2009) présente un planificateur qui calcule automatiquement des points de contacts qui vont aider le robot à réaliser des tâches d'atteinte.

3.5 Outils et logiciels utilisés dans cette thèse

Les algorithmes présentés dans les chapitres suivants ont été développés dans le cadre du projet *Humanoid Path Planner*, plateforme de développement logicielle dédiée

à la planification de mouvement pour robots humanoïdes. Cette plateforme s'appuie sur le logiciel KineoWorks ^{TM}(Laumond, 2006) qui implémente des algorithmes de planification de mouvement par échantillonnage. Le paquet logiciel `HPP-GIK` (pour *Generalized Inverse Kinematics*), dont une description précise peut être trouvée dans (Kanoun, 2009), implémente les algorithmes de cinématique inverse généralisée présentés en section 3.1 ainsi que le contrôle du ZMP présenté en section 3.2. Nos développements s'appuient sur ces logiciels.

Chapitre 4

Réduction de dimension en planification de mouvement

Ce chapitre présente un algorithme destiné à accélérer les planificateurs de mouvement dans la résolution de problèmes complexes. Le principe est d'utiliser en ligne les configurations libres de collision générées par le planificateur pour décrire l'espace libre. Étant donnée cette description, notre méthode accélère la progression du planificateur dans les directions identifiées comme favorables.

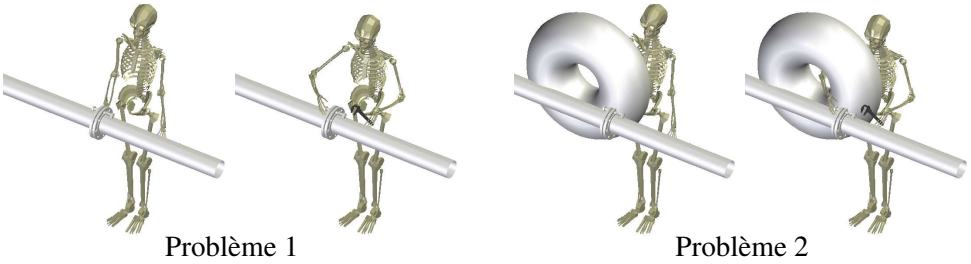
Nous présentons dans un premier temps les motivations de ce travail. Nous exposons ensuite une courte étude sur le ralentissement des algorithmes de planification de mouvement dans les « passages étroits » de $\mathcal{CS}_{\text{free}}$, puis la méthode statistique utilisée dans notre algorithme pour décrire la forme de $\mathcal{CS}_{\text{free}}$: l'analyse en composantes principales. Suivent une description de l'algorithme de planification lui-même et des résultats expérimentaux pour l'évaluer. Nous discutons à la fin du chapitre des limites et du champ d'application de notre méthode.

4.1 Motivations

Le travail présenté ici a été motivé par la planification de mouvement pour l'animation graphique. Dans ces problèmes hautement dimensionnés - nous nous intéressons à la planification corps complet, les planificateurs de mouvement classiques se comportent raisonnablement bien, mais peuvent être ralentis par la présence de passages étroits.

Toutefois, le fait que le mouvement du corps complet se déroule dans un passage étroit de $\mathcal{CS}_{\text{free}}$ ne traduit pas nécessairement la difficulté intrinsèque d'un problème de planification du point de vue d'un humain, comme nous allons le voir dans l'exemple suivant.

La figure 4.1 illustre l'exemple d'un personnage qui doit se mettre en position pour



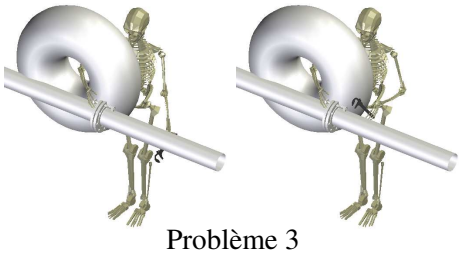


Figure 4.1 – Trois problèmes de planification de mouvement. Le squelette doit se mettre en position pour réparer un tuyau. Le premier problème est peu contraint, on ajoute un obstacle de forme torique pour les problèmes 2 et 3. Dans le problème 3, le bras droit du squelette est en position dès la position initiale, et le planificateur doit trouver un chemin sans collision pour mettre le bras gauche en position.

réparer un tuyau. Dans le premier problème, le personnage doit placer ses deux mains dans un environnement faiblement contraint. Ce problème n'est pas très difficile à résoudre pour un planificateur de mouvement classique. Dans le second problème, le personnage doit faire passer sa main droite à l'intérieur d'un tore pour se mettre en position. Ce nouvel obstacle rend la planification beaucoup plus difficile, car le chemin à trouver passe par un passage étroit de $\mathcal{CS}_{\text{free}}$. Le troisième et dernier problème se situe dans le même environnement que le deuxième, mais cette fois-ci, la main droite est déjà en position en configuration initiale. La difficulté de ce dernier problème, pour un humain, est plus ou moins équivalente à celle du premier problème : le personnage doit trouver un chemin pour son bras gauche dans un espace faiblement contraint. Toutefois, le fait que la main droite soit en position, et proche d'obstacles, force le planificateur à échantillonner des configurations dans un faible volume autour d'une sous-variété de $\mathcal{CS}$ (la sous-variété correspondant à un bras droit immobile). Cela ralentit les extensions aléatoires de $\mathcal{R}$, comme nous allons le quantifier dans la section suivante. Notre travail a été motivé par cette constatation, et a pour but d'identifier en ligne quand un processus de planification progresse dans un passage étroit.

Notons que nous souhaitons que cette identification se fasse de façon automatique, afin de conserver les propriétés de généricité des algorithmes de planification de mouvement. Cela signifie que dans le problème 3, par exemple, nous ne souhaitons pas qu'un

utilisateur donne au planificateur l'information sur les degrés de liberté à utiliser.

4.2 Complexité locale des méthodes d'expansion aléatoire

Le formalisme des espaces expansifs, présenté en section 2.4.1, décrit la complexité globale des espaces des configurations pour la planification de mouvement randomisée. Les résultats que nous allons présenter dans cette section visent à illustrer les propriétés géométriques locales qui ralentissent les planificateurs de mouvement.

Un planificateur de mouvement basé sur l'échantillonnage aléatoire tente de trouver un chemin dans $\mathcal{CS}$ entre une configuration initiale et finale avec aussi peu d'échantillons que possible. Cela signifie qu'un planificateur doit ajouter des arêtes dans $\mathcal{R}$ aussi longues que possible. Plus précisément, on souhaite échantillonner des configurations qui augmentent le plus possible le volume de la zone de visibilité de $\mathcal{R}$.

Soit q une configuration dans $\mathcal{CS}_{\text{free}}$. Pour comprendre comment un planificateur de mouvement va étendre $\mathcal{R}$ depuis q, il convient de considérer la zone visible depuis q : $V(q)$. Si $V(q)$ est également contrainte dans toutes les directions, une étape d'expansion de $\mathcal{R}$ depuis q donnera des résultats satisfaisants : on ajoutera une arête de longueur le rayon de $V(q)$, ce qui est le mieux qu'on puisse attendre. En revanche, si $V(q)$ a une forme allongée, on peut dire que q se trouve dans un passage étroit : une expansion isotrope depuis q sera ralentie en raison de contraintes selon certaines directions, alors qu'on pourrait progresser rapidement selon d'autres directions.

Ainsi, une bonne description de la complexité locale de $\mathcal{CS}$ est donnée par les variances et covariances de $V(q)$ selon les vecteurs d'une base de $\mathcal{CS}$: $\mathcal{R}$ peut être étendue selon les directions de plus grande variance, tandis que les contraintes selon les directions de faible variance ralentissent le planificateur.

4.2.1 Analyse d'une étape d'expansion à l'intérieur d'un passage étroit

D'après les remarques précédentes, ce qui caractérise un passage étroit est la forme allongée des régions visibles depuis les configurations à l'intérieur du passage. Pour dépasser cette description sommaire et quantifier le ralentissement des algorithmes de planification de mouvement à l'intérieur de passage, considérons l'exemple à deux dimensions représenté en Fig. 4.2. $\mathcal{CS}_{\text{free}}$ est localement de forme rectangulaire, de longueur constante égale à 1 et de largeur l tendant vers 0. On considère la configuration q au centre du rectangle, qu'on s'apprête à étendre. La configuration aléatoire vers laquelle on étend q est échantillonnée sur un cercle de centre q et de rayon $1/2$. Ainsi, l'expansion aléatoire depuis q est isotrope et la longueur maximale de l'extension est $1/2$.

Nous pouvons calculer explicitement la longueur moyenne $\overline{d}$ de cette extension. La

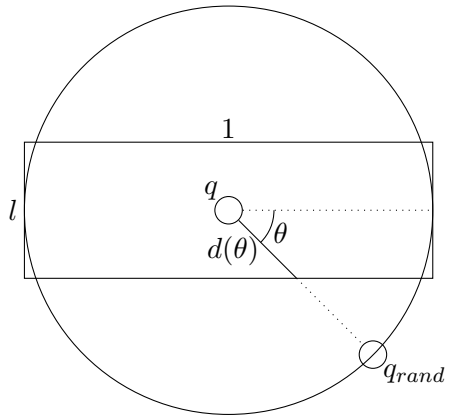


Figure 4.2 – Une extension aléatoire à l'intérieur d'un passage étroit. La longueur du rectangle est fixe et égale à 1. La largeur vaut l qui tend vers 0. Nous nous intéressons à la longueur moyenne $\overline{d}$ de la nouvelle arête de $\mathcal{R}$.

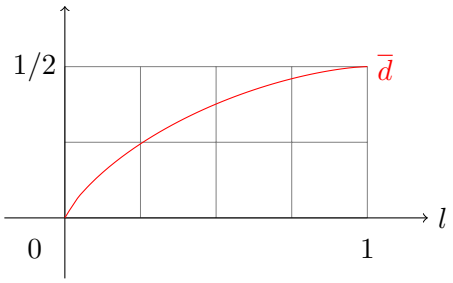


Figure 4.3 – Longueur moyenne d'une extension dans un passage étroit en fonction de l'épaisseur du passage.

direction de l'extension aléatoire est paramétrée par θ, par symétrie on ne considère que les $\theta \in [0, \pi/2]$. Pour $\theta \in [0, \arcsin(l)]$, $d(\theta) = 1/2$, pour $\theta \in [\arcsin(l), \pi/2]$, $d(\theta) = 1/2 \sin(\theta)$.

$$\begin{array}{rcl} \overline{d} & = & \frac{2}{\pi} \int_{\theta=0}^{\pi/2} d(\theta) d\theta \\ & = & \frac{1}{\pi} \left(\arcsin(l) - l \log(\tan(\arcsin(l)/2)) \right) \\ & \simeq_{l \to 0} & -\frac{l}{\pi} \log(l) \end{array}$$

Quand la largeur du rectangle tend vers 0, la longueur moyenne d'une extension tend vers 0, alors même que q voit des configurations éloignées dans l'une des deux directions. La figure 4.3 montre l'évolution de $\overline{d}$ en fonction de l.

Ce calcul simple peut être généralisé en dimension n. Soit q une configuration dans un passage dont p dimensions sont contraintes et $(n-p)$ sont libres. p est un entier entre 1 et $(n-1)$. On suppose toujours que la longueur des dimensions libres est constante égale à 1, et l'épaisseur du passage est l tendant vers 0.

Dans un repère centré en q, les coordonnées hypersphériques $(r, \boldsymbol{\theta})$ d'un point $(x_1, \ldots, x_n)$

sont telles que :

$$\begin{aligned} x_1 &= r\cos(\theta_1) \\ x_2 &= r\sin(\theta_1)\cos(\theta_2) \\ &\vdots \\ x_{n-1} &= r\sin(\theta_1)\sin(\theta_2)\dots\sin(\theta_{n-2})\cos(\theta_{n-1}) \\ x_n &= r\sin(\theta_1)\sin(\theta_2)\dots\sin(\theta_{n-2})\sin(\theta_{n-1}) \end{aligned}$$

Quand on étend q dans une direction $\boldsymbol{\theta}$, on obtient une extension de longueur $d(\boldsymbol{\theta})$. On cherche à calculer $\overline{d} = \frac{\int_{\boldsymbol{\theta}} d(\boldsymbol{\theta})J(\boldsymbol{\theta})d\boldsymbol{\theta}}{\int_{\boldsymbol{\theta}} J(\boldsymbol{\theta})d\boldsymbol{\theta}}$ où $J(\boldsymbol{\theta})$ est la jacobienne du changement de coordonnées hypersphériques : $J(\boldsymbol{\theta})d\boldsymbol{\theta} = \sin^{n-2}(\theta_1)\sin^{n-3}(\theta_2)\dots\sin(\theta_{n-2})d\theta_1\dots d\theta_{n-1}$. Par symétrie, on intégrera sur le domaine Ω tel que $\forall i \in [1, n-1], 0 \leq \theta_i \leq \pi/2$. Nous utiliserons plusieurs fois le fait que $\int_{\theta=0}^{\pi/2}\sin^n(\theta)d\theta = \frac{\sqrt{\pi}}{2}\frac{\Gamma((n+1)/2)}{\Gamma((n+2)/2)}$, où Γ est la fonction d'Euler.

Nous traiterons séparément le cas $(p = 1)$, qui est similaire à l'exemple en deux dimensions présenté au-dessus.

Cas $p = 1$

Nous utiliserons un lemme qui donne des primitives de $\frac{\sin^n(x)}{\cos(x)}$ pour $n \geq 0$.

Lemme 4.2.1. $\forall n \geq 0, \forall x \in [0, \frac{\pi}{2}[$, *Soit*

$$F_{2n}(x) = \sum_{i=0}^{n-1}\left(-\frac{1}{2i+1}\sin^{2i+1}(x)\right) + \log\left(\frac{1+\sin(x)}{\cos(x)}\right)$$

et

$$F_{2n+1}(x) = \sum_{i=1}^{n}\left(-\frac{1}{2i}\sin^{2i}(x)\right) - \log(\cos(x))$$

alors

$$\forall n \geq 0, \forall x \in [0, \frac{\pi}{2}[, F_n'(x) = \frac{\sin^n(x)}{\cos(x)}$$

Démonstration. Le résultat peut être vérifié par récurrence sur n. □

Supposons que x_1 est contrainte, c'est-à-dire localement $x_1 \leq l$. Alors $d(\boldsymbol{\theta})$ ne dépend que de θ_1 et sa valeur est donnée par :

$$\begin{cases} d(\boldsymbol{\theta}) = \frac{l}{\cos(\theta_1)} & \text{si } 0 \leq \theta_1 \leq \arccos(l), \\ d(\boldsymbol{\theta}) = 1 & \text{si } \arccos(l) \leq \theta_1 \leq \pi/2. \end{cases}$$

Ce qui donne :

$$\begin{aligned} \overline{d} &= \frac{\int_{\Omega} d(\boldsymbol{\theta})J(\boldsymbol{\theta})d\boldsymbol{\theta}}{\int_{\Omega} J(\boldsymbol{\theta})d\boldsymbol{\theta}} \\ &= \frac{\int_{\theta_1=0}^{\arccos(l)} l\frac{\sin^{n-2}(\theta_1)}{\cos(\theta_1)}d\theta_1 + \int_{\theta_1=\arccos(l)}^{\pi/2}\sin^{n-2}(\theta_1)d\theta_1}{\int_{\theta_1=0}^{\pi/2}\sin^{n-2}(\theta_1)d\theta_1} \end{aligned}$$

D'après le lemme 4.2.1,

$$\begin{array}{rcl} \int_{\theta_1=0}^{\arccos(l)} l\frac{\sin^{n-2}(\theta_1)}{\cos(\theta_1)}d\theta_1 & = & l\left(F_{n-2}(\arccos(l)) - F_{n-2}(0)\right) \\ & = & lF_{n-2}(\arccos(l)) \\ & \simeq_{l\to 0} & -l\log(l) \end{array}$$

De plus, $\forall \theta_1 \in [0, \pi/2], \sin^{n-2}(\theta_1) \leq \theta_1^{n-2}$, donc

$$\begin{array}{rcl} \int_{\theta_1=\arccos(l)}^{\pi/2} \sin^{n-2}(\theta_1)d\theta_1 & \leq & \frac{1}{n-1}(\pi/2)^{n-1}\left(1 - \left(\frac{\arccos(l)}{\pi/2}\right)^{n-1}\right) \\ & \simeq_{l\to 0} & (\pi/2)^{n-2}l \end{array}$$

et enfin $\int_{\theta_1=0}^{\pi/2} \sin^{n-2}(\theta_1)d\theta_1 = \sqrt{\pi}/2\Gamma((n-1)/2)/\Gamma(n/2)$. Donc on obtient, quand l tend vers 0 :

$$\overline{d} \simeq_{l\to 0} -\frac{2}{\sqrt{\pi}}\frac{\Gamma(n/2)}{\Gamma((n-1)/2)}l\log(l) \tag{4.1}$$

Cas $p \geq 2$

On suppose que $(x_{n-p+1}, \ldots, x_n)$ sont contraintes, c'est-à-dire localement $x_{n-p+1}^2 + \cdots + x_n^2 \leq l^2$. Alors, $d(\boldsymbol{\theta})\sin(\theta_1)\ldots\sin(\theta_{(n-p)}) \leq l$. La longueur $d(\boldsymbol{\theta})$ ne dépend que de $(\theta_1, \ldots, \theta_{(n-p)})$ et sa valeur est donnée par :

$$\left\{\begin{array}{rcl} d(\boldsymbol{\theta}) = & \frac{l}{\sin(\theta_1)\ldots\sin(\theta_{(n-p)})} & \text{si } \sin(\theta_1)\ldots\sin(\theta_{(n-p)}) \geq l, \\ d(\boldsymbol{\theta}) = & 1 & \text{si } \sin(\theta_1)\ldots\sin(\theta_{(n-p)}) \leq l \end{array}\right.$$

Soit V le domaine où $\sin(\theta_1)\ldots\sin(\theta_{(n-p)}) \leq l$ et S l'intégrale :

$$S = \int_\Omega \sin^{n-2}(\theta_1)\ldots\sin^{p-1}(\theta_{(n-p)})d\theta_1\ldots d\theta_{(n-p)}$$

On a :

$$\begin{array}{rcl} \overline{d} & = & \frac{\int_\Omega d(\boldsymbol{\theta})J(\boldsymbol{\theta})d\boldsymbol{\theta}}{\int_\Omega J(\boldsymbol{\theta})d\boldsymbol{\theta}} \\ & = & \left(\int_{\Omega - V} l\sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)})d\theta_1\ldots d\theta_{(n-p)} + \right. \\ & & \left. \int_V \sin^{n-2}(\theta_1)\ldots\sin^{p-1}(\theta_{(n-p)})d\theta_1\ldots d\theta_{(n-p)}\right)/S \\ & = & \left(\int_\Omega l\sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)})d\theta_1\ldots d\theta_{(n-p)}\right. \\ & & \left. + \int_V \sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)})\left(\sin(\theta_1)\ldots\sin(\theta_{(n-p)}) - l\right)d\theta_1\ldots d\theta_{(n-p)}\right)/S \end{array}$$

Pour $(\theta_1, \ldots, \theta_{(n-p)})$ dans V, on a $\sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)}) \leq l^{p-2}$, donc

$$\left|\int_V \sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)})\left(\sin(\theta_1)\ldots\sin(\theta_{(n-p)}) - l\right)d\theta_1\ldots d\theta_{(n-p)}\right| \leq \int_V l^{p-1}d\theta_1\ldots d\theta_{(n-p)}$$

De plus,

$$\left(\int_\Omega l\sin^{n-3}(\theta_1)\ldots\sin^{p-2}(\theta_{(n-p)})d\theta_1\ldots d\theta_{(n-p)}\right)/S = l\frac{\Gamma((p-1)/2)\Gamma(n/2)}{\Gamma(p/2)\Gamma((n-1)/2)}$$

d'où finalement

$$\overline{d} \simeq_{l \to 0} \frac{\Gamma((p-1)/2)\Gamma(n/2)}{\Gamma(p/2)\Gamma((n-1)/2)} l \tag{4.2}$$

Dans tous les cas, $\overline{d}$ tend vers 0 quand l tend vers 0. Comme on pouvait s'y attendre, cette convergence est plus rapide quand le nombre de dimensions contraintes augmente. Le résultat qualitatif important est que le ralentissement des planificateurs de mouvement à l'intérieur des passages étroits est dû à l'élongation des régions de visibilité, c'est-à-dire aux différences entres les variances de $\mathcal{CS}_{\text{free}}$ selon les directions de $\mathcal{CS}$.

La prochaine section présente une méthode statistique classique pour décrire de façon concise des nuages de points dans des espaces hautement dimensionnés dont les variances suivant les différentes dimensions varient.

4.3 Analyse en composantes principales (PCA)

L'analyse en composantes principales (notée PCA dans la suite) est un outil statistique de réduction de dimension. Étant donné un ensemble de p points dans un espace de dimension n, le but de la PCA est de déterminer si ces points peuvent être décrits par un sous-espace linéaire de dimension inférieure. La PCA a été imaginée par Pearson (Pearson, 1901), puis développée et formalisée par Hotelling (Hotelling, 1933). (Jolliffe, 2002) en présente une description exhaustive.

Description géométrique. Étant donné un ensemble de p points $(e_1, \ldots, e_p)$ centrés en 0 dans un espace euclidien de dimension n, on veut trouver la droite Δ qui approxime le mieux le nuage de points. Si on choisit comme critère que la somme des $d(e_i, \Delta)^2$ doit être minimale, la droite Δ est donnée par la direction selon laquelle la variance de $(e_1, \ldots, e_p)$ est maximale. Soit C la matrice de covariance des (e_i). C est symétrique et semi-définie positive. La variance des (e_i) le long d'une droite portée par un vecteur unitaire $\mathbf{u}$ est donnée par ${}^t\mathbf{u}C\mathbf{u}$. Le maximum de cette forme quadratique sous la contrainte ${}^t\mathbf{u}\mathbf{u} = 1$ est donnée par le vecteur propre u_1 associé à la plus grande valeur propre de C λ_1. Si on cherche à approximer le nuage de points par un plan, la deuxième direction du plan est donnée par le vecteur propre u_2, orthogonal à u_1, correspondant à la deuxième plus grande valeur propre de C λ_2, et ainsi de suite. La figure 4.4 illustre le calcul de la PCA sur un nuage de dix points en dimension deux.

L'algorithme 3 prend en entrée un ensemble de p points en dimension n et renvoie les matrices Λ et U contenant respectivement les valeurs propres et vecteurs propres de la matrice de covariance des points entrés.

4.4 Échantillonage controllé par la PCA

D'après l'étude de la section 4.2, un bon moyen de quantifier la complexité locale de $\mathcal{CS}$ est de décrire la forme des zones visibles depuis les points de $\mathcal{CS}_{\text{free}}$. À cette fin,

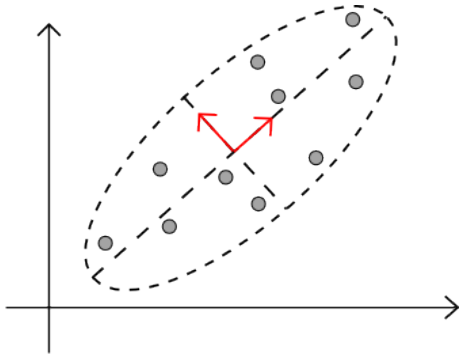


Figure 4.4 – Calcul de la PCA sur un nuage de dix points en dimension deux. Les deux vecteurs propres de la matrice de covariance sont représentés pas les flèches rouges.

Algorithme 3 PCA(Points$[n][p]$)

matrix C
for i = 1 to n **do**
  $C_{ii} \leftarrow$ Variance(Points$[i]$)
end for
for i, j = 1 to n, $i \neq j$ **do**
  $C_{ij} \leftarrow$ Covariance(Points$[i]$, Points$[j]$)
  $C_{ji} \leftarrow$ Covariance(Points$[i]$, Points$[j]$)
end for
matrix Λ, matrix U
$\Lambda, U \leftarrow$ EigenSolve(C)
return Λ, U

nous nous appuierons sur les résultats de la PCA sur des points échantillonés dans $\mathcal{CS}_{\text{free}}$. Suivant le paradigme de la planification de mouvement probabiliste, nous n'allons pas décrire explicitement $\mathcal{CS}_{\text{free}}$, mais plutôt utiliser en ligne les configurations libres de collisions de $\mathcal{R}$ obtenues par le planificateur de mouvement.

Étant donné un point de $\mathcal{CS}_{\text{free}}$ depuis lequel on s'apprête à effectuer une extension aléatoire, nous supposerons que ses plus proches voisins dans $\mathcal{R}$ décrivent correctement la forme locale de $\mathcal{CS}_{\text{free}}$. La PCA calculée sur ces points permettra de déterminer les directions dans $\mathcal{CS}$ selon lesquelles $\mathcal{CS}_{\text{free}}$ est le plus étendu. Suivant cette description, on modifiera la direction aléatoire de l'extension pour la faire suivre la forme de $\mathcal{CS}_{\text{free}}$. La figure 4.12 illustre la modification de la direction aléatoire d'une extension sur un algorithme de type RRT.

Notons que les voisins de q_{near} dans $\mathcal{R}$ qui sont à distance supérieure ou égale à deux ne sont pas nécessairement visibles depuis q_{near}. Nous les utiliserons cependant pour le calcul de la PCA. L'espace qu'on décrit est donc plutôt $\mathcal{CS}_{\text{free}}$ que $V(q_{near})$. On pourrait n'utiliser que des configurations effectivement visibles depuis q_{near}, mais cela nécessiterait plus d'itérations d'échantillonnage classique avant d'utiliser la PCA. En ef-

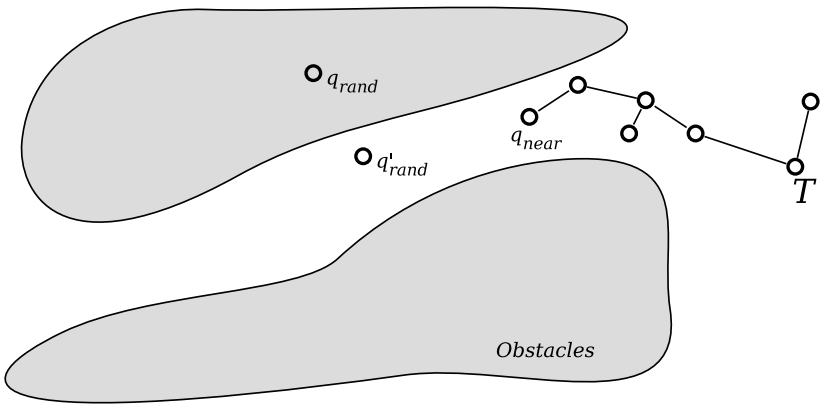


Figure 4.5 – Modification de la direction de l'extension aléatoire pour suivre la forme locale de $\mathcal{CS}_{\text{free}}$. Au lieu d'étendre q_{near} vers q_{rand}, on vise q'_{rand}.

fet, le calcul d'une PCA valide nécessite un nombre minimal de points. Nous discuterons en détail dans une prochaine section du nombre de points utilisés.

Une étape d'extension contrôlée par la PCA

Soit q une configuration de $\mathcal{R}$ qu'on s'apprête à étendre vers une configuration aléatoire q_{rand}. On trouve les p plus proches voisins de q dans $\mathcal{R}$ par un parcours en largeur, et on calcule une PCA sur ces points. Notons $\lambda_1 > \cdots > \lambda_n > 0$ les valeurs propres de leur matrice de covariance, et $\mathbf{u_1}, \ldots, \mathbf{u_n}$ les vecteurs propres correspondants. Dans un repère centré en q, on note $\left(q_{rand}^{(i)}\right)$ les coordonnées de q_{rand} dans la base propre. Alors, on transforme la direction aléatoire vers laquelle on étend $\mathcal{R}$ ainsi :

$$\forall i \in [1, n], q_{rand}^{\prime(i)} = \frac{\lambda_i}{\lambda_1} q_{rand}^{(i)}$$

Si on note U la matrice de changement de base vers la base propre, on peut écrire les coordonnées de q'_{rand} dans la base canonique de $\mathcal{CS}$:

$$q'_{rand} = q_{near} + \sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda_1} (q_{rand} - q_{near}).U_i \right) U_i$$

Cette transformation rapproche les coordonnées de q_{rand} de celles de q pour les directions de faibles variances tandis qu'elle laisse les coordonnées correspondant à des directions de variance élevée inchangées. L'algorithme 4 détaille une étape d'extension de $\mathcal{R}$ contrôlée par la PCA.

En reprenant l'exemple simple de la figure 4.2, nous pouvons montrer les différences entre une extension classique et une extension contrôlée par la PCA. La figure 4.6 montre comment une extension isotrope est transformée par la méthode décrite dans l'algorithme 4 en une ellipsoïde. La partie en rouge et gras indique représente la moitié de la probabilité pour la position de q_{rand} avant et après utilisation des résultats de la

Algorithme 4 PCA-Extend($q_{near}, q_{rand}, \mathcal{T}$)

vector table Points$[n][p]$
Points$\leftarrow$ BreadthFirstSearch($\mathcal{T}, q_{near}, p$)
matrix Λ, matrix U
$\Lambda, U \leftarrow$ PCA(Points)
$\lambda_{max} \leftarrow \max(\Lambda_{ii})$
$q_{rand} \leftarrow q_{near} + \sum_{i=1}^{n} \left(\frac{\Lambda_{ii}}{\lambda_{max}} (q_{rand} - q_{near}).U_i \right) U_i$
$q_{new} \leftarrow$ RRT-Extend(q_{near}, q_{rand})
return q_{new}

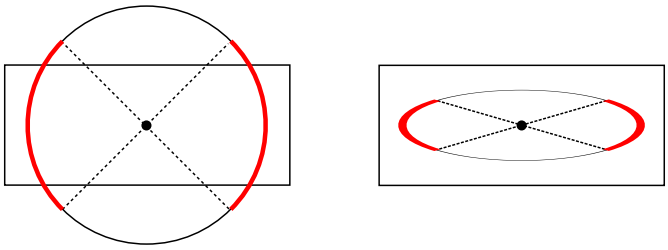


Figure 4.6 – Une extension aléatoire dans un passage étroit. À gauche, un échantillonnage isotrope et à droite un échantillonnage adapté par les résultats de la PCA, concentré suivant la direction du passage. La partie en rouge et gras représente la moitié de la mesure du tirage.

PCA. On peut voir que la probabilité se concentre sur les extrémités de l'ellipse quand on utilise `PCA-Extend()`. Ainsi, quand localement $\mathcal{CS}_{\text{free}}$ est plus étroit, la distance moyenne dont $\mathcal{R}$ est étendue ne converge pas vers 0, mais vers une limite strictement positive.

Les longueurs moyennes respectives d'une extension isotrope et transformée par PCA-Extend sont représentées en figure 4.7.

4.4.1 Précision du calcul de la PCA

Nous venons de décrire comment la connaissance de la forme locale de $\mathcal{CS}_{\text{free}}$ permettait d'accélérer les extensions aléatoires à l'intérieur d'un passage étroit. Pour valider notre approche, il reste à évaluer la précision de cette description. Nous calculons la PCA sur les plus proches voisins dans $\mathcal{R}$ d'une configuration de $\mathcal{CS}_{\text{free}}$; deux questions se posent : combien de voisins sont nécessaires et avec quelle précision la PCA calculée sur ces points décrit-elle $\mathcal{CS}_{\text{free}}$?

(Zwald & Blanchard, 2005) répond en partie à ces questions. Nous présentons ici une version simplifiée d'un théorème traitant de la vitesse de convergence de la PCA.

On considère une variable aléatoire X, qui prend ses valeurs dans un espace mesurable $\mathcal{X}$ suivant une distribution P. On a accès à p réalisations de X, sur lesquelles on effectue une PCA. On note C la matrice de covariance de X et C_p la covariance empi-

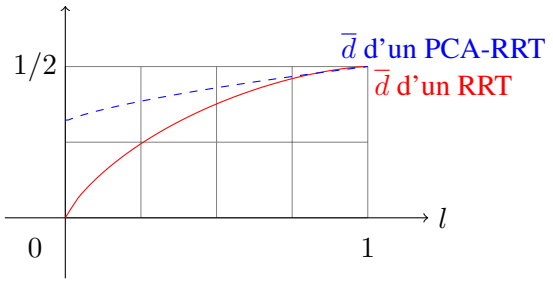


Figure 4.7 – Longueur moyenne d'une extension aléatoire à l'intérieur d'un passage étroit en fonction de l'étroitesse du passage. À droite, les deux dimensions sont également contraintes, et à gauche, l'épaisseur du passage tend vers 0. Alors que la longueur moyenne d'une extension de RRT classique (en trait plein rouge) tend vers 0, la longueur d'une extension contrôlée par la PCA (pointillés bleus) tend vers une limite strictement positive.

rique : mesurée après p tirages. On s'intéresse au sous-espace de dimension D engendré par les D vecteurs propres associés aux D plus grandes valeurs propres de C. On note ce sous-espace S_D, et la projection sur ce sous-espace P_{S_D}. Son équivalent empirique (obtenu à partir de C_p) est noté $\widehat{S_D}$ (respectivement $P_{\widehat{S_D}}$).

Les deux résultats présentés dans cette partie concernent :

– la vitesse de convergence de C_p vers C,
– la vitesse de convergence de $P_{\widehat{S_D}}$ vers P_{S_D}.

Remarque : Les résultats présentés dans (Zwald & Blanchard, 2005) sont plus généraux, et considèrent des opérateurs agissant dans un espace de Hilbert. La distance entre les opérateurs mesurés et réels utilise la norme de Hilbert-Schmidt. Nous sommes ici toujours en dimension finie et dans la suite, nous ne préciserons donc pas la norme utilisée dans les théorèmes. Les opérateurs de covariance ou de projection peuvent être assimilés à leur matrice dans la base canonique de $\mathcal{CS}$.

Le premier résultat est un lemme sur la convergence de la covariance empirique.

Lemme 4.4.1. *On suppose qu'il existe $M \in \mathbb{R}$ tel que pour tout $x \in \mathcal{X}$, $||x||^2 \leq M$. Alors, pour tout $\xi > 0$, avec probabilité supérieure à $1 - e^{-\xi}$,*

$$||C_p - C|| \leq \frac{2M}{\sqrt{p}} \left(1 + \sqrt{\frac{\xi}{2}}\right)$$

La condition de compacité de l'espace dans lequel on tire les points est toujours respectée dans les situations où nous utilisons la PCA. On observe une décroissance en $\frac{1}{\sqrt{(p)}}$.

Ce résultat est surtout présenté pour amener le théorème suivant. On donne ici une borne sur l'erreur faite dans l'estimation du sous-espace engendré par les D vecteurs propres correspondant aux D plus grandes valeurs propres.

Théorème 4.4.1. *On suppose qu'il existe $M \in \mathbb{R}$ tel que pour tout $x \in \mathcal{X}$, $||x||^2 \leq M$. Soient S_D et $\widehat{S_D}$ les sous-espaces engendrés par les D premiers vecteurs propres de C, respectivement C_p, comme définis précédemment. On note $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ les valeurs propres de C, pour $D > 0$, on pose $\delta_D = \frac{1}{2}(\lambda_D - \lambda_{D+1})$ et*

$$B_D = \frac{2M}{\delta_D}\left(1 + \sqrt{\frac{\xi}{2}}\right)$$

Alors, pour $p \geq B_D^2$, pour tout $\xi > 0$, avec probabilité supérieure à $1 - e^{-\xi}$,

$$\left|\left|P_{\widehat{S_D}} - P_{S_D}\right|\right| \leq \frac{B_D}{\sqrt{p}}$$

Ce résultat appelle deux remarques :

1. on retrouve la décroissance en $\frac{1}{\sqrt{(p)}}$ du lemme 4.4.1,
2. plus $\mathcal{CS}_{\text{free}}$ est étroit, c'est-à-dire plus $(\lambda_D - \lambda_{D+1})$ est élevé, plus les résultats de la PCA sont précis.

Notons que si l'espaces est également contraint dans toutes les directions (les (λ_i) sont proches les uns des autres), la borne du théorème 4.4.1 n'est pas intéressante. Dans ce cas, il se pourrait que la PCA renvoie des résultats imprécis. Toutefois, il s'agit précisément des cas où les algorithmes d'échantillonnage classiques fonctionnent correctement.

Remarque : Une des hypothèses du théorème ci-dessus est que les échantillons sont indépendants et identiquement distribués. Ce n'est pas le cas des nœuds d'un RRT par exemple, ça l'est pour des configurations tirées indépendamment et uniformément dans $\mathcal{CS}$ comme lors de la première phase de l'algorithme PRM. Nous présentons néanmoins ces résultats théoriques pour forger l'intuition sur la vitesse de convergence de la PCA. Le théorème sera par ailleurs utilisé comme un critère heuristique de convergence dans l'implémentation de l'échantillonnage contrôlé par la PCA, et ce indépendemment du fait que le théorème s'applique effectivement ou non. Les prochaines sections décrivent cette implémentation en détail.

4.4.2 Biais dans les espaces peu contraints

Le théorème 4.4.1 garantit que la PCA donne des résultats précis avec peu de points dans les passages étroits. En revanche, si $\mathcal{CS}_{\text{free}}$ est localement peu contraint - ou également contraint dans toutes les directions - la description renvoyée par la PCA peut

se révéler imprécise. Cela pourrait introduire un biais dans un planificateur de mouvement randomisé : si $\mathcal{CS}_{\text{free}}$ est peu contraint, l'algorithme favorisera les extensions dans des directions qui ont auparavant été explorées aléatoirement. Ce comportement est fortement indésirable pour la raison qu'on perd une des caractéristiques importantes des algorithmes de type RRT : la convergence de la distribution des nœuds de $\mathcal{R}$ vers la distribution des échantillons.

Pour s'affranchir de ce biais lorsque nous utilisons la PCA sur des algorithmes de type RRT, nous utilisons à chaque étape d'extension PCA-Extend avec probabilité $1/2$ et RRT-Extend avec probabilité $1/2$. Ainsi, $\mathcal{R}$ converge vers une couverture dense de $\mathcal{CS}_{\text{free}}$ tout en progressant plus vite dans les passages étroits.

4.4.3 Estimation automatique du nombre de points nécessaires au calcul de la PCA

L'algorithme 4 présenté précédemment contient un paramètre p : le nombre de points à utiliser pour décrire localement $\mathcal{CS}_{\text{free}}$. Si on se réfère au théorème 4.4.1, un choix correct pour ce nombre dépend de plusieurs facteurs : la dimension n de $\mathcal{CS}$, la densité des points utilisés et la forme locale de $\mathcal{CS}_{\text{free}}$, un passage plus étroit demandant moins de points pour être correctement identifié.

Nous avons implémenté une première version de PCA-RRT où p était fixé arbitrairement à $p = k.n$ pour des k variants entre 2 et 10, réglés par l'utilisateur en fonction du problème. Il faut en effet choisir $p \geq n + 1$ pour que la matrice de covariance des p points soit de rang plein. Cette solution a donné des résultats satisfaisants mais n'est pas acceptable du point de vue de la généricité de l'algorithme. Afin de déterminer *automatiquement* le nombre de points nécessaires à la description locale de $\mathcal{CS}_{\text{free}}$, nous avons choisi d'utiliser la borne du théorème 4.4.1 comme indicateur heuristique de la précision du résultat de la PCA.

Au lieu de décider à l'avance de p, on commence par calculer une PCA sur $(n + 1)$ points, puis on ajoute de nouveaux points un par un, en évaluant à chaque itération la précision du calcul. Quand le calcul atteint la précision désirée, on utilise les résultats pour transformer l'échantillon.

À chaque itération, étant donnés les valeurs propres $\lambda = (\lambda_1, \ldots, \lambda_n)$ de la matrice de covariance empirique, le nombre de points p utilisés pour les calculer et la distance maximale entre deux de ces points r, nous pouvons calculer la borne du théorème 4.4.1 pour tout D entre 1 et $(n - 1)$. Soit $f_D(\lambda, p, r) = \frac{4r^2}{\sqrt{p}(\lambda_D - \lambda_{D+1})}$.

La norme du projecteur sur un sous-espace linéaire de dimension D est $\sqrt{D}$. En reprenant les notations du théorème 4.4.1, posons $\delta = \left|\left| P_{\widehat{S_D}} - P_{S_D} \right|\right|$. L'erreur relative faite lors de l'évaluation de S_D est $\delta/\sqrt{D}$.

Pour $\epsilon > 0$ donné, si $f_D(\lambda, p, r)/\sqrt{D} \leq \epsilon/1.707$ (en choisissant $\xi = 1$ dans le théorème 4.4.1), on obtient, avec probabilité au moins $1 - e^{-1}$, $\delta/\sqrt{D} \leq \epsilon$. Dans notre implémentation, nous avons choisi comme critère de convergence $\epsilon = 0.1$: s'il existe

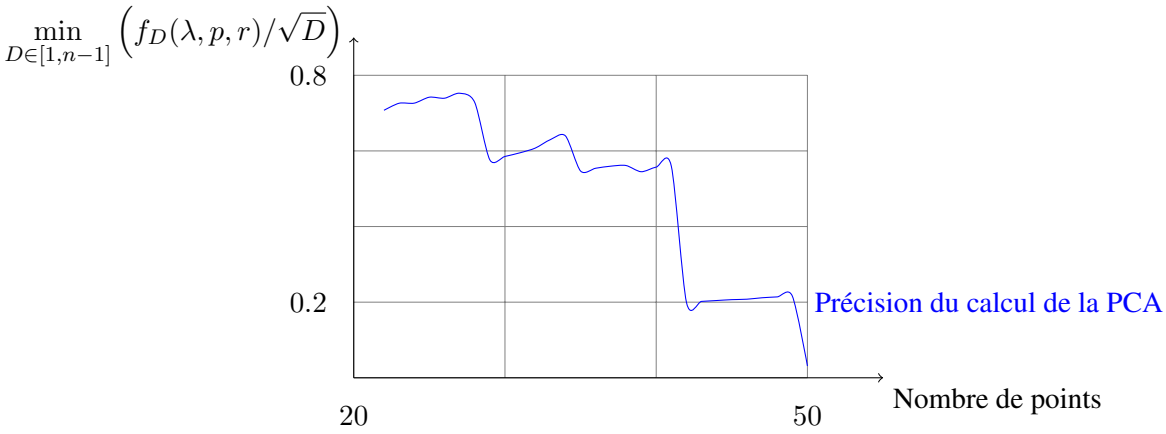


Figure 4.8 – Convergence de l'estimation d'une borne supérieure de la distance entre le sous-espace linéaire qui approxime $\mathcal{CS}_{\text{free}}$ et le sous-espace identifié empiriquement par la PCA. Le calcul s'arrête lorsque cette estimation passe sous un certain seuil, ce qui est le cas ici pour cinquante points. Les données sont issues d'une exécution de PCA-RRT sur le problème présenté en figure 4.1.

$D \in [1, n-1]$ tel que $f_D(\lambda,p,r)/\sqrt{D} \leq 0.059$, on arrête d'ajouter des points pour décrire $\mathcal{CS}_{\text{free}}$ et on utilise les résultats renvoyés par la PCA.

La figure 4.8 montre un exemple de calculs successifs de PCA pour un robot à vingt degrés de liberté. La condition d'arrêt est atteinte lors du calcul de PCA sur cinquante points de $\mathcal{CS}_{\text{free}}$. Ces données sont issues d'une exécution de l'algorithme PCA-RRT sur le problème présenté en figure 4.1.

PCA récursive

Le problème de cette méthode est qu'elle nécessite de recalculer une PCA à chaque fois qu'on désire prendre un point de plus en compte. Heureusement, il existe dans la littérature des descriptions de PCA incrémentales moins couteuses que la mise à jour de la matrice de covariance de l'ensemble des points puis sa diagonalisation. L'algorithme que nous présentons est tiré de (Erdogmus *et al.*, 2004). Il s'agit d'une méthode pour approximer au rang 1 la mise à jour des valeurs propres et vecteurs propres de la matrice de covariance en fonction d'un nouveau point à prendre en compte.

On suppose dans la suite que les données sur lesquelles on effectue une PCA sont centrées, le calcul dans le cas non centré diffère peu. Supposons qu'on a réalisé une PCA sur k points $(\mathbf{x_i})_{i=1..k}$. On veut mettre à jour ce calcul après la mesure de $\mathbf{x_{k+1}}$. Notons C_k la matrice de covariance des $(\mathbf{x_i})_{i=1..k}$. On peut écrire la relation de récurence :

$$C_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} \mathbf{x_i}\mathbf{x_i}^T = \frac{k}{k+1}C_k + \frac{1}{k+1}\mathbf{x_{k+1}}\mathbf{x_{k+1}}^T \tag{4.3}$$

On a déjà diagonalisé C_k, et on souhaite diagonaliser C_{k+1}. Notons $C_k = Q_k \Lambda_k Q_k^T$,

Q est la matrice de changement de base vers la base propre et Λ la matrice diagonale des valeurs propres de C. Notons enfin $\mathbf{y_{k+1}} = Q_k^T \mathbf{x_{k+1}}$. C'est le nouveau point exprimé dans la base propre de l'ancienne covariance.

En réinjectant ces notations dans 4.3, on obtient :

$$Q_{k+1}\Lambda_{k+1}Q_{k+1}^T = Q_k \left[\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T \right] Q_k^T \tag{4.4}$$

Il suffit donc de diagonaliser $\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T$, pour obtenir Q_{k+1} et Λ_{k+1}. Posons :

$$\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T = VDV^T$$

avec D diagonale et V orthonormale, on obtient :

$$\begin{cases} Q_{k+1}{=}Q_kV \\ \Lambda_{k+1}{=}\ D \end{cases}$$

La matrice à diagonaliser a une structure particulière : c'est la somme d'une matrice diagonale et d'une matrice de norme petite devant celle de la matrice diagonale. Le calcul qui suit montre comment la diagonaliser en faisant des approximations de rang 1 sur $\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T$.

Analyse de perturbations sur une matrice diagonale On veut diagonaliser $(\Lambda + \alpha\alpha^T)$ avec $\alpha\alpha^T$ petit devant Λ. On note $(\Lambda + \alpha\alpha^T) = VDV^T$. Comme la matrice à diagonaliser est proche de Λ, on pose $D = \Lambda + P_\Lambda$ et $V = I + P_V$: les valeurs propres de $(\Lambda + \alpha\alpha^T)$ sont proches des valeurs propres de Λ et ses vecteurs propres aussi. P_Λ et P_V sont petites devant Λ, respectivement I. En développant VDV^T, on obtient :

$$\begin{aligned} VDV^T{=}& \qquad (I+P_V)(\Lambda+P_\Lambda)(I+P_V)^T \\ {=}&\Lambda + P_\Lambda + DP_V^T + P_VD + P_V\Lambda P_V^T + P_VP_\Lambda P_\Lambda^T \end{aligned}$$

Les termes $P_V\Lambda P_\Lambda^T$ et $P_VP_\Lambda P_V^T$ sont négligeables (ordre 2 en (P_V, P_Λ)). En identifiant $VDV^T = \Lambda + \alpha\alpha^T$, on obtient :

$$\alpha\alpha^T = P_\Lambda + DP_V^T + P_VD \tag{4.5}$$

Par ailleurs, V est orthogonale, ce qui se traduit, pour P_V, par $(I+P_V)(I+P_V)^T = I$. Le terme $P_VP_V^T$ est négligeable, ce qui donne : $P_V = -P_V^T$.

On obtient donc les coefficients de P_Λ et P_V en fonction de α grâce à 4.5. Notons λ_i le $i^{\text{ème}}$ élément diagonal de Λ, et α_i la $i^{\text{ème}}$ coordonnée de α, on a :

$$\begin{cases} (P_\Lambda)_{ij}{=} & 0 & (\text{si } i \neq j) \\ (P_\Lambda)_{ii}{=} & \alpha_i^2 & \\ (P_V)_{ij}{=} & \frac{\alpha_i\alpha_j}{\lambda_j+\alpha_j^2-\lambda_i-\alpha_i^2} & (\text{si } i \neq j) \\ (P_V)_{ii}{=} & 0 & \end{cases}$$

4.5 Complexité

La complexité de l'algorithme à chaque étape d'expansion de $\mathcal{R}$ ne dépend que de deux paramètres : la dimension de $\mathcal{CS}$ n et le nombre de points utilisés pour calculer la PCA p.

Le premier calcul de PCA sur $(n+1)$ points est fait en $O(n^3)$ opérations. Ensuite, chaque étape de calcul de PCA récursive comprend un calcul de produit de matrices $n \times n$, donc la complexité totale est $O((p-n) \times n^3)$. Nous n'avons malheureusement aucun moyen de prédire la valeur de p, qui dépend de chaque système et de chaque problème. Nous présenterons dans la prochaine section des résultats expérimentaux qui incluront la valeur moyenne de p et les coûts relatifs de calcul de PCA et de détection de collision.

Remarque : L'algorithme 3 correspond à l'implémentation naïve de la description mathématique de la PCA, *via* le calcul de la matrice de covariance des points entrés. C'est cette implémentation que nous avons utilisée dans les résultats présentés dans la suite du chapitre. Un meilleur choix pourrait être d'utiliser la décomposition en valeurs singulières (*Singular Value Decomposition* ou SVD en anglais) (Wall *et al.*, 2003) de la matrice de données. Cette solution serait plus robuste numériquement dans le cas où les plus petites valeurs propres de la matrice de covariance sont proches de 0. De nombreuses méthodes ont été proposées pour calculer des versions incrémentales de SVD, par exemple (Bunch & Nielsen, 1978), ou plus récemment (Brand, 2002). Le coût de la mise à jour d'une SVD quand on ajoute un point est à notre connaissance, toujours $O(n^3)$ si la matrice de données est de rang plein (n) et qu'on désire conserver toutes les composantes principales, comme c'est le cas dans notre application.

La complexité algorithmique totale serait donc la même que celle présentée ici, si on désire conserver la version incrémentale de l'algorithme PCA-RRT qui détermine automatiquement le nombre de points nécessaires. En effet, le coût de la première SVD sur $(n+1)$ points est $O(n^3)$, puis la mise à jour des résultats à chaque ajout de point se fait en $O(n^3)$, on a donc toujours un coût total en $O((p-n) \times n^3)$.

4.6 Résultats expérimentaux

Dans cette section, nous comparons notre méthode d'échantillonage controlée par la PCA à plusieurs algorithmes randomisés de planification de mouvement.

Toutes les expériences présentées dans cette section ont été réalisées sur un PC Intel Core 2 Duo 2,13 GHz avec 2 Go de RAM. Chaque méthode a été lancée cent fois sur chaque problème. Nous présentons les résultats moyens de ces essais. Pour chaque problème, les configurations initiales et finales définissant le problème sont fixes. Nous rapportons le temps de calcul moyen, son écart-type, le nombre d'itérations des algorithmes testés et le nombre de configurations valides générées (nœuds de $\mathcal{R}$). Nous avons par

ailleurs indiqué la longueur moyenne des extensions (dans une unité arbitraire), avec une méthode d'échantillonnage classique et avec l'échantillonnage contrôlé par la PCA.

Pour chaque essai, nous avons arrêté le planificateur après un million d'itérations s'il n'avait pas trouvé de solution. Nous avons indiqué le pourcentage d'essais infructueux pour chaque problème. Notons que plus ce pourcentage est haut, plus le temps de calcul moyen mesuré est une sous-estimation du temps de calcul moyen réel (sans jamais arrêter le planificateur).

L'utilisation la plus évidente de notre méthode est de transformer un algorithme de planification qui repose uniquement sur des extensions aléatoires de $\mathcal{R}$: l'algorithme RRT.

4.6.1 Problème de désassemblage mécanique

L'environnement présenté en figure 4.9 est l'arrière d'une voiture. Le système considéré est le tuyau d'échappement, qu'on souhaite démonter. Ce problème est fourni avec KineoWorks^{TM}. La configuration initiale du robot est en position montée, le planificateur doit trouver un chemin jusqu'à la position démontée. La planification se déroule dans un passage étroit de $\mathcal{CS}_{\text{free}}$: en position montée, le pot d'échappement ne peut presque pas bouger. La figure 4.9 illustre le problème et présente un chemin solution.

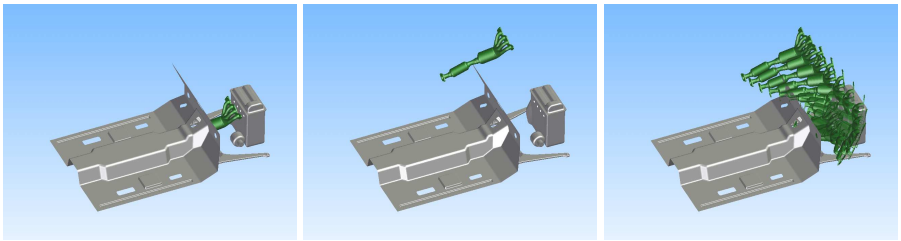


Figure 4.9 – Problème de désassemblage d'un pot d'échappement. Configurations initiale et finale et chemin solution.

Le tableau 4.1 présente les résultats moyens des algorithmes RRT et PCA-RRT sur ce problème. La PCA a utilisé en moyenne 14.5 points pour décrire localement $\mathcal{CS}_{\text{free}}$. L'algorithme PCA-RRT a passé 1% du temps de calcul à calculer la PCA et 93% aux détections de collisions. Comme la dimension de $\mathcal{CS}$ est faible (6), la PCA coûte peu à chaque itération. Cependant, le gain à utiliser la PCA pour biaiser l'échantillonnage n'est pas élevé, car la réduction de dimension aide peu dans un espace de dimension 6. Dans les exemples suivants, nous verrons que la PCA coûte beaucoup plus comparée aux détections de collisions, mais se traduit par un gain plus important en termes de nombre d'itérations et de temps de calcul.

	Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne
RRT	7748,3	5241,2	272,239 s	87,855 s	96,5
PCA-RRT	4725,2	3362,2	196,405 s	82,978 s	118,3

Table 4.1 – Résultats expérimentaux pour le problème de désassemblage d'un pot d'échappement.

4.6.2 Problèmes d'animation graphique

L'algorithme présenté ici a été pensé pour pallier les limitations des algorithmes randomisés classiques de planification de mouvement. Les problèmes à haute dimension sont généralement difficiles pour les planificateurs de mouvement en raison du nombre exponentiel attendu de configurations nécessaires pour décrire $\mathcal{CS}_{\text{free}}$. Les problèmes présentés en figure 4.1 traitent d'un robot possédant un nombre plutôt élevé de degrés de liberté : il s'agit du haut du corps d'une figure anthropomorphe. Le personnage peut bouger son torse, sa tête et ses bras, ce qui représente dans le modèle utilisé vingt degrés de liberté. Le but dans ces problèmes est de se mettre en position pour réparer un tuyau. Il tient une clé à molette dans sa main gauche et sa main droite doit tenir un écrou. Pour cela, son bras droit doit passer dans le trou d'un gros obstacle en forme de tore. Le tore est composé de 1 000 triangles. Les difficultés dans cet environnement proviennent de l'espace étroit dans lequel le bras droit doit passer, de la dimension totale de $\mathcal{CS}$ et des collisions causées par les mains et la clé à molette. Nous allons considérer les problèmes présentés en figure 4.1 dans l'ordre suivant :

- D'abord le problème où le bras droit du personnage est déjà en position. Le planificateur de mouvement doit trouver un chemin autour d'un sous-espace linéaire de $\mathcal{CS}$ qui correspond à un bras droit immobile.
- Ensuite le problème où le personnage doit bouger ses deux bras. La partie la plus difficile vient du bras droit qui doit suivre le passage étroit constitué par le trou du tore. Le passage à suivre dans $\mathcal{CS}_{\text{free}}$ n'est pas linéaire.

De même que pour le problème précédent, nous nous intéressons ici à l'intérêt d'utiliser la PCA pour accélérer les extensions aléatoires à l'intérieur d'un passage étroit, et pas pour trouver l'entrée du passage. Pour tous ces problèmes, nous avons donc choisi d'enraciner les arbres de recherche dans la configuration finale (les deux mains sont en position pour réparer le tuyau), c'est-à-dire à l'intérieur du passage étroit. Ainsi, le planificateur n'a qu'à suivre le passage.

Bras droit positionné

Pour la première famille de problèmes, la main droite est positionnée à l'intérieur du trou pour les configurations initiales et finales. Le planificateur doit trouver un chemin pour placer la main gauche. $\mathcal{CS}_{\text{free}}$ est un petit volume autour d'un sous-espace linéaire de $\mathcal{CS}$.

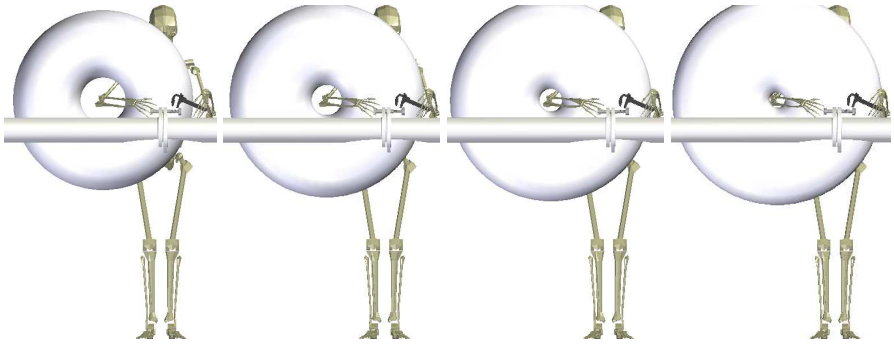


Figure 4.10 – Différents scénarios de difficulté croissante due à l'étroitesse du trou dans lequel doit passer le bras droit.

Afin d'évaluer comment les avantages à utiliser la PCA dépendent de l'étroitesse des passages dans $\mathcal{CS}_{\text{free}}$, nous avons fait varier la taille du trou dans lequel se trouve la main droite. Dans la première expérience, nous n'avons pas ajouté le tore. Ensuite, la taille du trou dans lequel le bras droit doit passer est successivement 24 cm, 16 cm, 12 cm et 8 cm. Les configurations initiales et finales sont les mêmes pour tous ces problèmes. La figure 4.10 présente les différents environnements, du plus facile au plus contraint.

La figure 4.11 présente un chemin solution pour ce problème. Les résultats des algorithmes RRT et PCA-RRT sont présentés dans les tables 4.2 et 4.3. Le nombre moyen de points utilisés par la PCA pour décrire $\mathcal{CS}_{\text{free}}$ localement vaut 31,2. Ce nombre varie légèrement entre 28,3 pour le problème le plus facile et 32,6 pour le plus difficile. Pour ces problèmes, le calcul de la PCA représente 80% du temps de calcul total, le reste étant principalement consacré aux détections de collisions. Ce taux ne varie pas significativement au cours des différents problèmes. L'algorithme PCA-RRT a toujours trouvé une solution pour tous ces problèmes. L'algorithme RRT classique, en revanche, a souvent échoué à trouver une solution en moins d'un million d'itérations. Nous avons indiqué en table 4.2 le taux d'échec de l'algorithme. Rappelons que plus ce taux est haut plus les temps de calcul et nombre d'itérations moyens mesurés sous-estiment les valeurs moyennes réelles.

Nous pouvons observer que l'algorithme PCA-RRT est plus efficace que l'algorithme RRT sur chacun des problèmes. Il trouve une solution plus vite, en moins d'itérations, et échoue moins souvent. Une remarque intéressante est que plus le passage à trouver dans $\mathcal{CS}$ est étroit, plus le gain moyen à utiliser la PCA est élevé. La figure 4.12 montre l'évolution du rapport entre la longueur moyenne des extensions dues à un échantillonnage contrôlé par la PCA et de celles dues à un échantillonnage classique. Si on se réfère à l'analyse illustrée par la figure 4.4, on doit attendre de ce rapport qu'il diverge vers $+\infty$ quand l'étroitesse du passage tend vers 0.

Figure 4.11 – Chemin solution pour la première famille de problèmes. Le personnage doit bouger son bras gauche pour se mettre en position.

Taille du trou	Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne	Taux d'échec
Pas de tore	56 446	182,2	123,302 s	455,952 s	1,23	4%
24 cm	240 691	188,7	514,678 s	850,086 s	0,335	20%
16 cm	254 300	166,0	528,041 s	861,391 s	0,255	22%
12 cm	303 732	177,7	686,534 s	1 019,66 s	0,165	23%
8 cm	305 794	164,4	709,301 s	1 010,62 s	0,0836	26%

Table 4.2 – Résultats moyens de l'algorithme RRT sur la première famille de problèmes de planification pour l'animation graphique.

Taille du trou	Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne
Pas de tore	1 104	111,8	4,377 s	3,097 s	50,7
24 cm	2 970	166,6	14,881 s	11,531 s	9,65
16 cm	5 111	200,0	27,795 s	28,418 s	7,72
12 cm	6 588	198,6	34,679 s	42,311 s	5,11
8 cm	11 533	177,9	50,992 s	53,595 s	2,70

Table 4.3 – Résultats moyens de l'algorithme PCA-RRT sur la première famille de problèmes de planification pour l'animation graphique.

Mouvement des deux bras

L'environnement est le même que pour les problèmes précédents, mais cette fois-ci le personnage doit placer ses deux mains pour réparer le tuyau. La figure 4.13 présente un chemin solution pour ce problème.

Nous avons lancé le RRT classique cent fois sur le problème où le bras droit doit passer par un trou de 24 cm, et il n'a jamais trouvé de solution en moins d'un million d'itérations. Nous n'avons donc pas pensé qu'il était pertinent de lancer les expériences sur les environnements plus contraints, comme pour le problème précédent. Les résultats

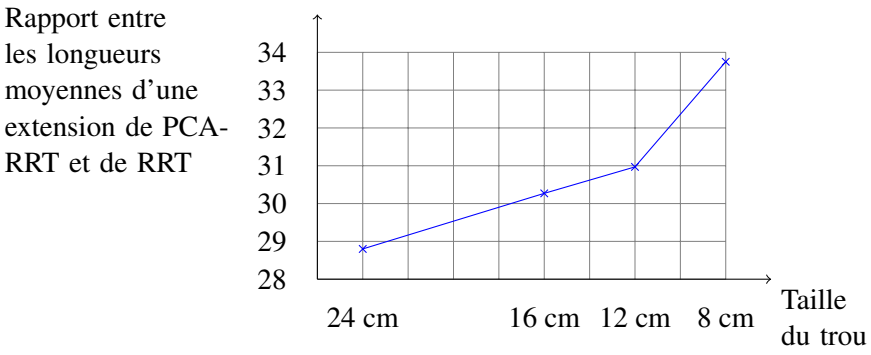


Figure 4.12 – Évolution du rapport entre la longueur moyenne d'une extension contrôlée par la PCA et d'une extension classique de l'algorithme RRT. Quand le trou devient plus petit, le passage dans $\mathcal{CS}_{\text{free}}$ devient plus étroit, et le gain à utiliser la PCA s'accroît.

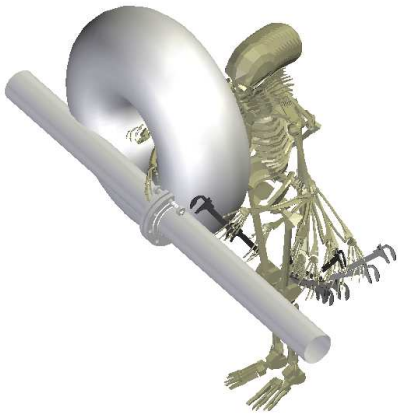


Figure 4.13 – Chemin solution pour la seconde famille de problèmes d'animation graphique. Le personnage doit bouger ses deux bras pour se mettre en position.

moyens des algorithmes RRT et PCA-RRT sont présentés respectivement en tables 4.2 et 4.3. La PCA a utilisé en moyenne 33,2 points pour décrire localement $\mathcal{CS}_{\text{free}}$. Son coût est 70% du temps de calcul total. L'écart type du temps de calcul pour le RRT classique pour ce problème n'est pas intéressant. Sa faible valeur s'explique par le fait que l'algorithme a été systématiquement arrêté après le même nombre d'itérations. Pour ce problème, le rapport entre la longueur moyenne des extensions dues à l'échantillonnage contrôlé par la PCA et de celles dues à un échantillonnage classique monte jusqu'à 300.

Problème d'animation en environnement industriel

Ce problème nous a été posé par Dassault Aviation dans le cadre du projet ANR-RNTL PerfRV2. La photographie en figure 4.14 et le modèle 3D correspondant sont utilisés avec leur accord.

Taille du trou	Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne	Taux d'échec
Pas de tore	257 173	306,9	622,947 s	993,628 s	0,43	22%
24 cm	1 000 000	333,7	2 168,174 s	250,409 s	0,058	100%

Table 4.4 – Résultats moyens de l'algorithme RRT sur la seconde famille de problèmes de planification pour l'animation graphique.

Taille du trou	Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne	Taux d'échec
Pas de tore	1 901	165,4	8,026 s	5,373 s	27	0%
24 cm	137 982	2 638,9	1 055,062 s	926,660 s	18	18%

Table 4.5 – Résultats moyens de l'algorithme PCA-RRT sur la seconde famille de problèmes de planification pour l'animation graphique.

Dans ce problème, deux personnages doivent coopérer afin de monter une pièce sur une soute d'avion. Il est facile pour le personnage du dessus de se mettre en position. En revanche, le personnage sous la soute évolue dans un environnement très contraint. Le système considéré ici est constitué des degrés de liberté du haut du corps du personnage sous la soute. Comme dans les problèmes d'animation précédents, il consiste en 20 degrés de liberté. La figure 4.14 expose le problème : à gauche la configuration initiale et à droite la configuration finale. La figure 4.15 présente un chemin solution.

Les résultats des algorithmes RRT et PCA-RRT sont présentés en tables 4.6 et 4.7. Comme pour le problème précédent, $\mathcal{CS}_{\text{free}}$ n'est pas simplement un sous-espace linéaire de $\mathcal{CS}$. L'utilisation de la PCA se révèle tout de même utile, à cause de la présence de passages étroits. La PCA a utilisé en moyenne 32,4 points pour décrire localement $\mathcal{CS}_{\text{free}}$. Les calculs de PCA représentent pour ce problème 88% du temps de calcul total.

4.6.3 L'algorithme PRM contrôlé par la PCA

Afin d'étudier plus largement l'utilité de contrôler l'échantillonnage aléatoire par la PCA en planification de mouvement, nous avons implémenté d'autres planificateurs que le RRT. L'algorithme PRM tel que décrit dans (Kavraki *et al.*, 1996) comporte deux phases :

- d'abord un échantillonnage uniforme de $\mathcal{CS}$, au cours duquel on conserve toutes les configurations libres de collision, qu'on essaye de connecter entre elles,
- puis un échantillonnage adaptatif autour des nœuds déjà obtenus dont on soupçonne qu'ils se trouvent à l'intérieur de passages étroits.

L'heuristique proposée dans (Kavraki *et al.*, 1996) pour déterminer si une configuration q se trouve à l'intérieur d'un passage étroit consiste à compter le nombre de fois où le planificateur a essayé de relier q à une autre configuration sans y parvenir. La probabilité de choisir q pour une extension aléatoire dans la deuxième phase de l'algorithme est proportionnelle à ce nombre. Nous avons suivi cette heuristique dans notre

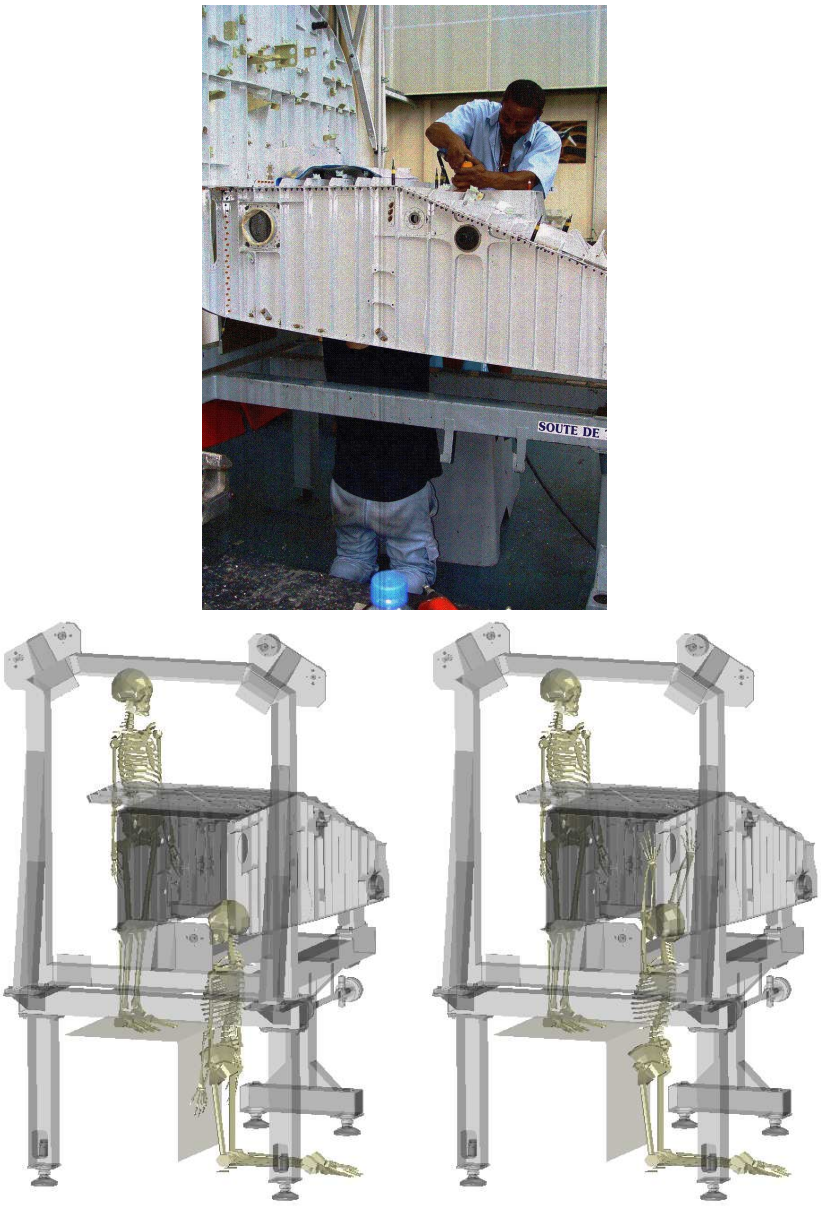


Figure 4.14 – Problème industriel de planification de mouvement pour un acteur digital. La difficulté principale est de placer les deux mains du personnage sous la soute. La photographie illustrant le problème et le modèle 3D de la soute sont utilisés avec l'autorisation de Dassault Aviation. Le problème nous a été proposé dans le cadre du projet ANR-RNTL PerfRV2.

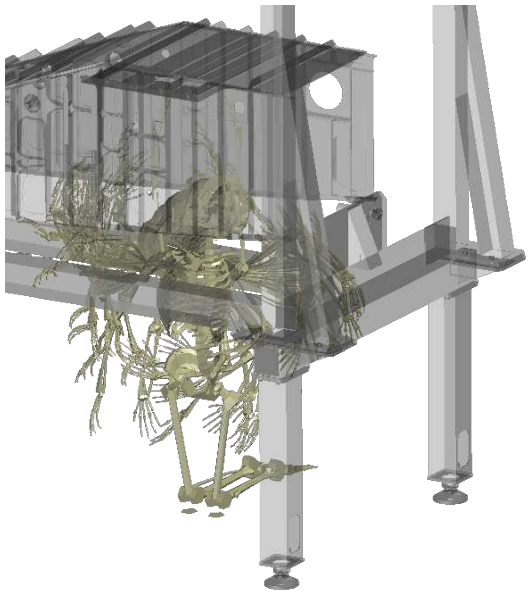


Figure 4.15 – Chemin solution pour le problème de la soute d'avion.

Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne	Taux d'échec
870 852	1 424	2 484,212 s	786,087 s	0,0402	72%

Table 4.6 – Résultats moyens de l'algorithme RRT sur le problème industriel de planification pour l'animation graphique.

Itérations	Nœuds	Temps de calcul moyen	Écart-type	Longueur moyenne	Taux d'échec
186 769	2 602	1 406,818 s	868,773 s	1,07	34%

Table 4.7 – Résultats moyens de l'algorithme PCA-RRT sur le problème industriel de planification pour l'animation graphique.

implémentation de l'algorithme PRM.

L'algorithme PRM a été conçu pour construire un plan de $\mathcal{CS}_{\text{free}}$ puis répondre à de multiples requêtes de planification. Nous avons choisi de suivre l'implémentation proposée dans (Hsu *et al.*, 2006) pour l'adapter à des requêtes simples : on itère la première phase 100 fois, puis la deuxième 200 fois et on recommence, jusqu'à trouver un chemin entre les configurations initiale et finale.

Le contrôle de l'échantillonnage par la PCA peut se faire durant la deuxième phase. Lorsqu'un nœud est choisi pour une extension aléatoire, on réalise une PCA sur ses voisins dans $\mathcal{R}$. Si le calcul de la PCA converge, nous utilisons les résultats pour transformer la direction de l'extension.

Nous présentons les résultats de PRM et PCA-PRM sur le problème du pot d'échappement dans les tables 4.8 et 4.9. Les algorithmes PRM et PCA-PRM ont systématique-

Itérations	Nœuds	Temps de calcul moyen	Écart-type
11 957	9 423	869,541 s	1 542,159 s

Table 4.8 – Résultats moyens de l'algorithme PRM sur le problème de désassemblage mécanique.

Itérations	Nœuds	Temps de calcul moyen	Écart-type
7 688	6 973	490,557 s	798,127 s

Table 4.9 – Résultats moyens de l'algorithme PCA-PRM sur le problème de désassemblage mécanique.

ment trouvé une solution au problème en moins d'un million d'itérations.

Pour ce problème (comme pour ceux d'animation présentés précédemment), la première phase d'échantillonnage uniforme de l'algorithme PRM n'est pas aussi efficace qu'un algorithme RRT qui ne réalise que des extensions aléatoires. La PCA permet tout de même d'accélérer la phase d'extension de $\mathcal{R}$.

4.6.4 Iterative Path Planning (IPP) contrôlé par la PCA

Iterative Path Planning (IPP) est un algorithme de planification de mouvement reposant sur des stratégies d'expansion d'arbres aléatoires. Il a été présenté dans (Ferre & Laumond, 2004). Il s'agit de calculer une solution approximative en exécutant l'algorithme RRT tout en autorisant au robot de « pénétrer » dans les obstacles. Ce chemin est ensuite remodelé itérativement en réduisant la pénétration autorisée et en relançant des RRT. L'idée qui sous-tend cet algorithme est que les premières exécutions - rapides - de l'algorithme RRT vont servir à trouver les entrées des passages étroits. Une fois ces entrées trouvées, les RRT avec une faible pénétration vont pouvoir raffiner les solutions approximatives rapidement. Il a été observé que cet algorithme est très efficace pour résoudre des problèmes de désassemblage mécanique. IPP fait partie du logiciel KineoWorks^{TM}, nous avons donc utilisé son implémentation. Nous avons adapté l'échantillonnage de la même façon que pour les RRT.

Nous présentons les résultats moyens de IPP et PCA-IPP sur le problème de désassemblage du tuyau d'échappement dans les tables 4.10 et 4.11. IPP résout ce problème plus rapidement que PCA-RRT. Ici, l'heuristique qui consiste à agrandir les passages étroits est plus efficace que celle qui consiste à linéariser le passage. Il reste intéressant de coupler les deux heuristiques, ainsi PCA-IPP est légèrement plus rapide que IPP.

Dans les problèmes hautement dimensionnés tels que les problèmes d'animation présentés précédemment, l'heuristique d'IPP n'est pas toujours aussi efficace. Les tables 4.12 et 4.13 présentent les résultats des algorithmes IPP et PCA-IPP sur le problème d'animation dans lequel le personnage doit bouger son bras gauche, tandis que son bras droit se trouve à l'intérieur d'un tore. La version du problème considérée est celle où le trou du tore a une taille de 24 cm. PCA-IPP a systématiquement trouvé un chemin solution, on a indiqué dans la table le taux d'échec de IPP.

Itérations	Nœuds	Temps de calcul moyen	Écart-type
212	42,2	3,543 s	1,731 s

Table 4.10 – Résultats moyens de l'algorithme IPP sur le problème de désassemblage mécanique.

Itérations	Nœuds	Temps de calcul moyen	Écart-type
185	40,2	3,324 s	1,381 s

Table 4.11 – Résultats moyens de l'algorithme PCA-IPP sur le problème de désassemblage mécanique.

Itérations	Nœuds	Temps de calcul moyen	Écart-type	Taux d'échec
74 097	357,3	309,634 s	1 135,415 s	2%

Table 4.12 – Résultats moyens de l'algorithme IPP sur le problème d'animation graphique.

Itérations	Nœuds	Temps de calcul moyen	Écart-type
20 134	87,1	67,066 s	111,421 s

Table 4.13 – Résultats moyens de l'algorithme PCA-IPP sur le problème d'animation graphique.

Dans cet exemple, IPP est plus efficace que RRT. PCA-IPP et PCA-RRT requièrent quant à eux moins de temps de calcul qu'IPP. On remarque enfin que PCA-IPP est plus lent que PCA-RRT. Les avantages à utiliser IPP dépendent pour beaucoup de la longueur des passages étroits à traverser dans $\mathcal{CS}_{\text{free}}$. Si un passage est long, comme c'est le cas ici, de nombreuses configurations échantillonnées à l'intérieur du passage valides pour une certaine pénétration sont invalidées quand le seuil de pénétration autorisée est baissé. Cela induit de nombreux calculs inutiles. Si en revanche le passage est ponctuel, comme pour le problème de désassemblage présenté précédemment, les premiers appels à l'algorithme RRT peuvent localiser l'entrée du passage, ce qui revient quasiment à la résolution du problème.

Dans tous les cas, on observe que l'échantillonnage contrôlé par la PCA est plus efficace que l'échantillonnage classique. La combinaison d'IPP et du contrôle de l'échantillonnage par la PCA n'est toutefois pas systématiquement plus efficace que PCA-RRT.

4.7 Discussion sur le choix de la technique de réduction de dimension

4.7.1 Méthodes locales non-linéaires

La PCA est une méthode de réduction de dimension globalement linéaire. Étant donnée une configuration q dans $\mathcal{CS}_{\text{free}}$ depuis laquelle on s'apprête à effectuer une extension, la PCA sert à décrire succintement la forme des faisceaux de trajectoires libres de collision qui partent de q. Si les trajectoires élémentaires (telles que construites

par la méthode locale utilisée) ne sont pas des lignes droites, il est vain d'essayer de décrire leur ensemble par la PCA. C'est le cas par exemple pour les robots qui possèdent des contraintes dynamiques ou non-holonomes comme les voitures. Pour ces systèmes, la méthode décrite précédemment n'est pas pertinente.

Il existe dans la littérature des méthodes de réduction de dimension non-linéaires dont l'analyse en composantes principales à noyau (Kernel PCA en anglais, notée KPCA dans la suite) (Schoelkopf *et al.*, 1997). Étant donné un ensemble de points, la KPCA réalise une PCA sur une représentation de ces points dans un espace de plus grande dimension. Cette technique générique est appelée en apprentissage automatique le « kernel trick ». Elle est utilisable car le calcul de la PCA ne dépend que du produit scalaire entre les points d'entrée pris deux à deux. Notons ϕ la fonction qui permet de passer de $\mathcal{CS}$ à l'espace dans lequel on souhaite réaliser la PCA - il n'est pas nécessaire pour l'algorithme KPCA d'avoir une écriture explicite de ϕ. Le calcul de la PCA sur l'image de l'ensemble de points par ϕ ne dépend que des $\phi(x).\phi(y)$. Le principe du *kernel trick* est d'accéder à ces valeurs via une fonction noyau k, définie ici sur $\mathcal{CS}^2$ telle que pour $(x, y) \in \mathcal{CS}^2, \phi(x).\phi(y) = k(x, y)$.

Prenons l'exemple d'un robot voiture, dont la méthode locale génère des courbes de Reeds et Shepp (Reeds & Shepp, 1990). On peut associer une distance d à cette méthode locale telle que la distance entre deux configurations q_1 et q_2 $d(q_1, q_2)$ soit la longueur de la courbe de Reeds et Shepp qui relie q_1 à q_2. Étant donnée cette distance, on peut définir ce que serait le produit scalaire correspondant dans un espace de plus haute dimension, dans lequel l'image de ces courbes sont des droites : $k(q_1, q_2) = \left(d(0, q_1)^2 + d(0, q_2)^2 - d(q_1, q_2)^2\right)/2$. Nous n'avons pas montré qu'un tel k définissait bien un produit scalaire. Nous l'avons toutefois utilisé pour contrôler l'échantillonnage d'un algorithme de planification de mouvement utilisé sur un robot qui décrit des courbes de Reeds et Shepp. Pour de tels systèmes, KPCA-RRT se comporte mieux que PCA-RRT. Toutefois, sur ces problèmes à faible dimension - 3 pour une voiture dont la configuration est décrite par (x, y, θ) - il n'est pas très intéressant d'avoir recours à des méthodes de réduction de dimension, et dans les exemples que nous avons étudié un simple RRT était généralement plus efficace qu'à la fois PCA-RRT et KPCA-RRT.

4.7.2 Description non-linéaire de $\mathcal{CS}_{\text{free}}$

Dans le travail présenté ici, nous avons choisi d'utiliser la PCA comme technique de réduction de dimension car elle est facile à implémenter et fournit de bons résultats. La principale limitation de cette méthode est qu'elle est globalement linéaire. Les sous-espaces renvoyés par la PCA, qui sont censés décrire $\mathcal{CS}_{\text{free}}$ dans l'utilisation qu'on en fait, sont ainsi linéaires et pourraient être de dimension supérieure qu'une sous-variété non-linéaire approximant correctement $\mathcal{CS}_{\text{free}}$.

Il existe dans la littérature des méthodes de réduction de dimension non-linéaires, parmi lesquelles ISOMAP (Tenenbaum *et al.*, 2000) ou « Locally Linear Embedding » (Roweis & Saul, 2000). Plusieurs cas de figures peuvent se présenter quand on utilise la

PCA pour contrôler l'échantillonnage de planificateurs de mouvement :

1. La sous-variété à identifier est effectivement linéaire. C'est le cas dans le problème d'animation présenté précédemment où le bras droit du personnage est placé correctement dès la configuration initiale. Dans ce problème, certains degrés de liberté ne doivent pas être modifiés (ceux du bras droit), ce qui force le planificateur à échantillonner des configurations sur une intersection d'hyperplans, c'est-à-dire un sous-espace linéaire de $\mathcal{CS}$. Dans ce cas, il est pertinent d'utiliser la PCA comme outil de réduction de dimension.
2. La forme locale de $\mathcal{CS}_{\text{free}}$ n'est pas linéaire, mais lisse. Puisque notre utilisation de la PCA est locale, nous allons identifier l'espace tangent à la sous-variété qui approxime $\mathcal{CS}_{\text{free}}$. La PCA projette alors les extensions aléatoires dans les directions tangentes à la sous-variété ce qui accélère la résolution du problème. Les problèmes d'animation ou de désassemblage mécanique en sont de bonnes illustrations. Il pourrait toutefois être intéressant de décrire non-linéairement $\mathcal{CS}_{\text{free}}$ pour de l'échantillonnage statique, comme lors de la première phase de l'algorithme PRM.
3. $\mathcal{CS}_{\text{free}}$ prend un virage à angle droit. Dans ce cas, la direction renvoyée par la PCA n'aide en rien la résolution du problème. Lorsqu'utilisée avec l'algorithme RRT, les étapes d'échantillonnage classique garantissent que le planificateur va échantillonner des configurations après le virage. Une fois que l'arbre de recherche a passé le virage, les deux directions orthogonales vont être considérées par la PCA, qui n'en favorisera pas une par rapport à l'autre.
4. $\mathcal{CS}_{\text{free}}$ a une forme de « T ». L'utilisation de la PCA induit une inertie dans la direction en ligne droite. Si le chemin à trouver prend le virage à angle droit, on peut dire que l'utilisation de la PCA ralentit l'algorithme, en ce sens qu'elle va favoriser l'exploration dans la mauvaise direction.

4.8 Conclusion

Nous avons présenté dans ce chapitre une adaptation des algorithmes randomisés de planification de mouvement. La méthode statistique utilisée, la PCA, est connue et largement utilisée dans la littérature, mais pas encore pour contrôler l'échantillonnage aléatoire en planification de mouvement. Son utilisation et l'analyse de sa validité, dans ce cadre, sont les contributions de notre travail. Comme espéré, nous avons observé que la PCA pouvait accélérer la progression des graphes de recherche dans des espaces de configurations fortement contraints. Nous n'avons pas résolu le problème qui consiste à trouver l'entrée des passages étroits dans $\mathcal{CS}_{\text{free}}$, mais seulement celui de la progression à l'intérieur de ces passages.

Pour valider notre méthode, nous l'avons utilisée sur des problèmes variés, représentant différents champs d'application de la planification de mouvement. Un des principaux atouts de la technique proposée est qu'elle ne nécessite aucun paramétrage :

l'algorithme s'adapte automatiquement à la dimension et à la difficulté des problèmes considérés. La classe de problèmes pour laquelle notre algorithme s'est révélé le plus utile est l'animation de personnages possédant de nombreux degrés de liberté et évoluant dans des environnements complexes.

Chapitre 5

Planification de mouvement sous contraintes

Le chapitre précédent a présenté une méthode pour adapter les algorithmes de planification aux systèmes hautement dimensionnés. La planification de mouvement appliquée aux robots humanoïdes se heurte à une difficulté autre que la grande dimension de $\mathcal{CS}$: les configurations et chemins qu'on génère doivent respecter des contraintes d'équilibre, de faisabilité physique ou encore de tâches dans l'espace opérationnel du robot. La figure 5.1 présente un exemple de tâche de manipulation (un mouvement d'atteinte) pendant laquelle on souhaite que le robot conserve son équilibre tout en évitant les collisions avec les obstacles.

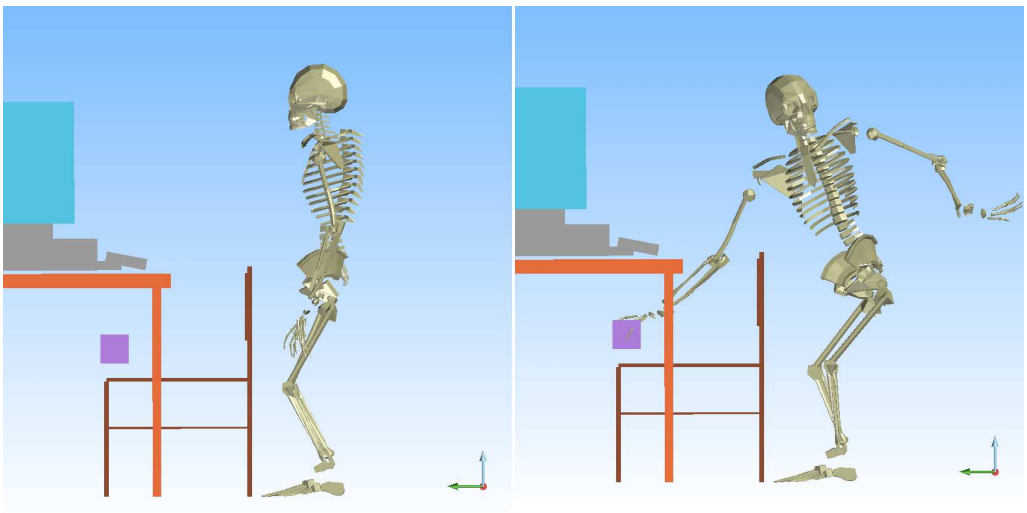


Figure 5.1 – Un mouvement d'atteinte dans un environnement contraint. Le robot réalise la tâche de manipulation tout en conservant son équilibre.

Ce chapitre présente une méthode originale pour utiliser les algorithmes randomisés de planification de mouvement conjointement avec des méthodes de cinématique inverse qui garantissent le respect de contraintes entrées par l'utilisateur, ou la réalisation de tâches de manipulation.

5.1 Utilisation de la cinématique inverse

Ce travail utilise intensivement le formalisme de cinématique inverse présenté en 3.1 pour résoudre différents types de tâches, parmi lesquelles :

1. l'équilibre statique : Pour un robot humanoïde se tenant sur ses deux jambes, l'équilibre statique d'une configuration peut être exprimé simplement par une tâche sur le centre de masse. Au cours d'un mouvement statiquement stable, on requerra typiquement que les deux pieds ne bougent pas et que le centre de masse se projette au centre du polygone support du robot dans le plan du sol.
2. la réalisation d'un but : les problèmes de manipulation s'expriment souvent comme une tâche à réaliser plutôt que comme une configuration à atteindre. Nous présenterons en particulier des exemples où le robot doit atteindre un certain point ou une certaine orientation avec ses mains.
3. l'exploration de $\mathcal{CS}$: comme nous l'expliquerons plus en détail dans les sections suivantes, nous utilisons des tâches dont la valeur s'exprime dans $\mathcal{CS}$ pour implémenter des algorithmes d'exploration aléatoire de $\mathcal{CS}$.

5.2 Planification de mouvement corps-complet pour un robot humanoïde

Cette section présente une adaptation des algorithmes randomisés de planification de mouvement à la planification de chemins articulaires pour un robot humanoïde. Les algorithmes de cinématique inverse hiérarchisée servent à garantir des propriétés d'équilibre des mouvements planifiés, tandis que l'architecture globale inspirée des algorithmes de type RRT permet d'éviter les obstacles.

La conception d'algorithmes de planification de mouvement sous contraintes pose un problème principal : l'échantillonnage de sous-variétés de $\mathcal{CS}$ de dimension inférieure à $\mathcal{CS}$. Un échantillonnage uniforme de $\mathcal{CS}$, par exemple, génère des configurations valides avec une probabilité nulle. Les algorithmes de cinématique inverse hiérarchisée présentés dans la section 3.1 servent à pallier ces limites.

Quitte à utiliser des algorithmes de résolution de tâches de cinématique inverse, il serait possible d'y intégrer des contraintes d'évitement d'obstacles. Cette intégration a été présentée dans (Kanehiro *et al.*, 2008a; Kanehiro *et al.*, 2008b). Nous présenterons à la fin de cette section une comparaison entre notre stratégie et la leur.

5.2.1 Échantillonnage de sous-variétés de $\mathcal{CS}$

Le problème étudié ici est celui de la génération de configurations aléatoires qui appartiennent à une sous-variété donnée de $\mathcal{CS}$. Les sous-variétés considérées sont définies implicitement comme les solutions d'ensembles de tâches. Nous ne disposons donc pas d'une paramétrisation des variétés à explorer qui permettrait d'échantillonner directement l'espace des paramètres.

Afin de générer des configurations aléatoires satisfaisant un ensemble de tâches passé en entrée, nous utilisons les algorithmes de cinématique inverse présentés dans la section 3.1. La méthode de planification de mouvement randomisée qui consiste à générer des configurations uniformément distribuées dans $\mathcal{CS}$ puis à vérifier que ces configurations appartiennent à $\mathcal{CS}_{\text{free}}$ devient donc :

1. Générer une configuration q dans $\mathcal{CS}$ avec une loi de probabilité uniforme,
2. Projeter q pour satisfaire les tâches T , on note $p(T, q)$ le projeté de q sur la sous-variété vérifiant T.
3. Vérifier que $p(T, q)$ est libre de collision.

Les problèmes de manipulation pour robots humanoïdes sont souvent posés comme des tâches à résoudre dans $\mathcal{WS}$ plutôt que des configurations à atteindre dans $\mathcal{CS}$. Cette méthode permet de se ramener à la seconde forme en échantillonnant la sous-variété solution de la tâche but jusqu'à générer suffisament de configurations finales. La figure 5.2 présente quatre configurations libres de collisions, générées aléatoirement, qui satisfont des contraintes d'équilibre et une tâche d'atteinte de la main droite.

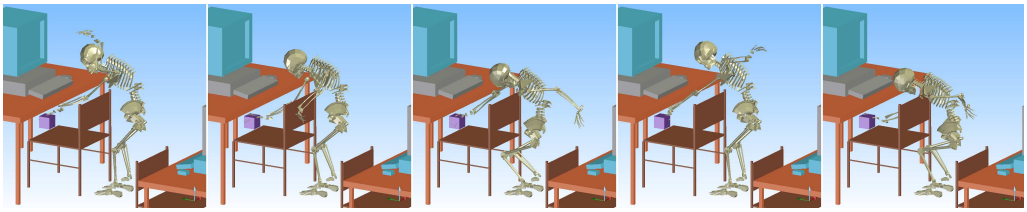


Figure 5.2 – Échantillonnage de sous-variété de $\mathcal{CS}$. Les contraintes à vérifier sont : les positions des pieds, la position du centre de masse et la position de la main droite. $\mathcal{CS}$ est échantillonné uniformément, chaque échantillon est ensuite projeté sur la sous-variété qui vérifie les contraintes grâce aux outils de cinématique inverse, puis passé au détecteur de collisions.

On peut adapter les algorithmes de planification en appliquant cette méthode à toutes les configurations générées aléatoirement. Nous avons remarqué qu'il était plus efficace de réécrire les méthodes d'extension pour y intégrer les algorithmes de cinématique inverse. C'est ce que présente la prochaine section.

5.2.2 Extensions sous contraintes dans $\mathcal{CS}$

Afin d'adapter efficacement les algorithmes de type RRT à l'exploration de sous-variétés contraintes de $\mathcal{CS}$, nous avons choisi d'implémenter spécifiquement une méthode d'extension sous contraintes dans $\mathcal{CS}$. Partant d'une configuration q_{near} qui respecte un ensemble de contraintes $\mathcal{C}$, on désire se rapprocher autant que possible d'une configuration tirée au hasard dans $\mathcal{CS}$: q_{rand}, tout en restant sur la sous-variété $\mathcal{V}$ vérifiant $\mathcal{C}$. Pour ce faire, nous utilisons les algorithmes de cinématique inverse avec priorités.

Définissons la tâche $T^{(q_0)}_{\text{config}}$, dite *de configuration*, indexée par une configuration q_0 par :

$$\forall q \in \mathcal{CS}, T^{(q_0)}_{\text{config}}(q) = (q - q_0)$$

En ajoutant cette tâche à la plus faible priorité dans la pile des contraintes et tâches, on génère, grâce aux algorithmes de cinématique inverse avec priorités, la solution locale au problème d'optimisation :

$$q_{\text{solution}} = \underset{q \in \mathcal{V}}{\operatorname{argmin}} \left(||q - q_0|| \right)$$

Lors d'une extension sous contraintes dans $\mathcal{CS}$, nous utilisons cette tâche avec comme configuration cible, les configurations qu'un algorithme RRT classique aurait ajoutées à $\mathcal{R}$. Chaque configuration résultat de la cinématique inverse est ensuite passée au détecteur de collisions, ainsi que les chemins élémentaires dans $\mathcal{CS}$ reliant les configurations entre elles. Les figures 5.3 et 5.4 présentent une comparaison entre une étape d'extension d'un RRT classique et une étape d'extension sous contraintes.

L'algorithme 5 présente le pseudo-code correspondant à une étape d'extension sous contraintes. La fonction `LocalPlanner.Solve()` correspond à un appel aux algorithmes de cinématique inverse avec priorités.

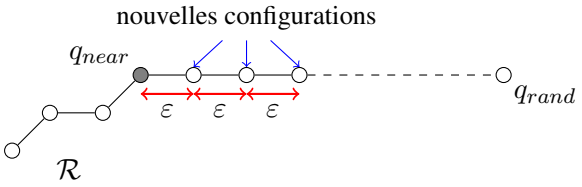


Figure 5.3 – Une étape d'extension de l'algorithme RRT-Connect

5.2.3 Optimisation de posture

Comme lors de l'utilisation d'algorithmes de planification classiques, les chemins résultats, générés aléatoirement, doivent être optimisés. Le critère à optimiser dépend du système considéré. En robotique humanoïde, on souhaite généralement que le mouvement final semble « réaliste » ou « naturel », pour une définition satisfaisante de ces notions.

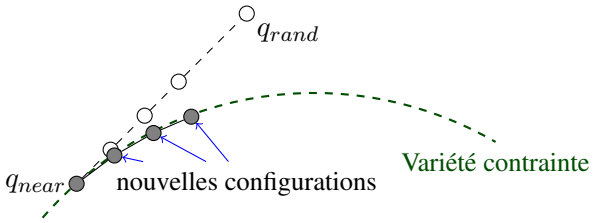


Figure 5.4 – Une étape d'extension sous contraintes

Algorithme 5 Constrained-Connect($\mathcal{R}$, q_{near}, q_{rand})

$Tasks$.Initialize($Constraints$)
$Tasks$.Add($ConfigurationTask$)
$\Delta_q \leftarrow \varepsilon.(q_{rand} - q_{near})/||(q_{rand} - q_{near})||$
$q_{target} \leftarrow q_{near} + \Delta_q$
$q_{current} \leftarrow q_{near}$
$State \leftarrow$ Progressing
while $State =$ Progressing **do**
  $ConfigurationTask$.SetTarget(q_{target})
  $q_{new} \leftarrow$ LocalPlanner.Solve($Tasks$, $q_{current}$)
  if $q_{new} \neq q_{current}$
  and CollisionCheck(q_{new}) = OK
  and CollisionCheckEdge($q_{current}$, q_{new}) = OK
  and ConstraintsCheck(q_{new}) = OK **then**
    $\mathcal{R}$.AddNode(q_{new})
    $\mathcal{R}$.AddEdge($q_{current}$, q_{new})
    $q_{current} \leftarrow q_{new}$
    if $q_{target} \neq q_{rand}$ **then**
      $q_{target} \leftarrow q_{target} + \Delta_q$
    else
      $State \leftarrow$ Reached
    end if
  else
    $State \leftarrow$ Trapped
  end if
end while

Dans l'algorithme présenté ici, la configuration finale du chemin solution a été générée aléatoirement, et pas entrée par un utilisateur. Pour que le mouvement semble naturel, il est donc important d'optimiser non seulement le chemin solution, mais aussi la posture finale atteinte par l'algorithme de planification. Notons que nous n'optimisons une configuration finale qu'après l'avoir atteinte plutôt que d'optimiser les configurations finales dès qu'elles sont générées, ce afin d'éviter des calculs inutiles, l'optimisation de posture étant l'étape la plus coûteuse de notre processus de planification, comme nous l'observerons dans la partie expérimentale de cette section.

L'optimisation de posture se déroule comme suit : partant d'une configuration finale valide et sans collision, et étant donnée une fonction objectif à minimiser, le planificateur génère des déplacements élémentaires aléatoires qui minimisent la fonction objectif tout en vérifiant les contraintes (y compris la contrainte définissant les configurations finales). On trouve ainsi un chemin sans collision entièrement inclus dans la sous-variété but. Dans les exemples présentés ici, le critère qu'on a choisi d'optimiser est la distance à une configuration de référence dans $\mathcal{CS}$. La figure 5.5 présente les étapes d'une optimisation de posture. Les contraintes appliquées au robot sont l'équilibre statique (deux pieds au sol, centre de masse projeté au milieu du polygone support) et la tâche de but qui consiste à placer la main droite sur la boîte violette posée sur la table. Les configurations générées sont de plus en plus naturelles.

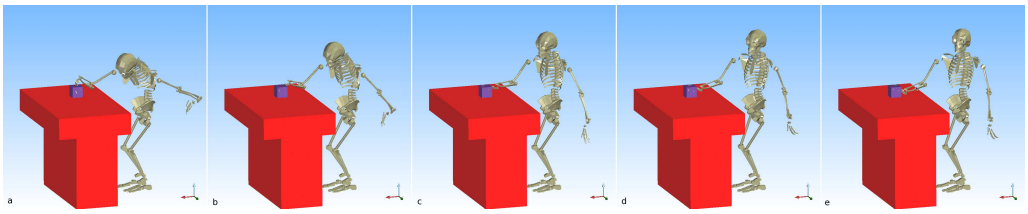


Figure 5.5 – Étapes d'une procédure d'optimisation de posture. Toutes les configurations sont sans collision, respectent les contraintes d'équilibre et la tâche sur la position de la main. De gauche à droite, les configurations sont de plus en plus naturelles.

Le nouveau chemin solution est la concaténation du premier chemin solution et du chemin d'optimisation de la posture finale. On peut ensuite optimiser ce chemin solution avec les méthodes classiques d'optimisation de trajectoires de la planification de mouvement randomisée.

5.2.4 Architecture globale de l'algorithme

Maintenant que nous avons présenté les différentes étapes de la planification de mouvement sous contraintes, résumons l'architecture globale qui permet de résoudre un problème de planification.

Étant donnée une pile de tâche qu'on souhaite résoudre,

1. On génère plusieurs configurations finales libres de collision qui résolvent les tâches,
2. On lance un algorithme RRT, utilisant la fonction `Constrained-Connect()` pour générer les nouvelles configurations aléatoires.
3. Quand un chemin a été trouvé entre la configuration initiale et une configuration finale, on optimise cette dernière,
4. On optimise la concaténation du chemin aléatoire solution et du chemin d'optimisation de posture.

5.3 Comparaison avec une méthode locale d'évitement d'obstacles

Cette section présente des résultats expérimentaux obtenus en simulation avec le modèle de robot HRP-2. Nous avons comparé en détails notre algorithme avec la méthode locale d'évitement de collisions présentée dans (Kanehiro *et al.*, 2008a). Nous avons choisi de présenter des comparaisons de ces méthodes sur deux exemples publiés de tâches d'atteinte de la main.

Le premier exemple est un mouvement où les deux pieds sont au sol, le robot humanoïde doit atteindre un objet situé sous une table. Le second est un mouvement plus compliqué, le robot se tient sur un seul pied et doit passer son bras droit à travers un large obstacle torique. Des obstacles gênent aussi les mouvements du bras gauche et de la jambe en suspension.

Notre algorithme utilise les planificateurs de mouvement et les détecteurs de collision de KineoWorks. Toutes les simulations ont été réalisées sur un PC Intel Core 2 Duo 2,13 GHz avec 2 Go de RAM. Les temps de calculs présentés dans (Kanehiro *et al.*, 2008a; Kanehiro *et al.*, 2008b) ont été obtenus sur la même machine, ce qui rend les comparaisons de temps de calculs pertinentes. Pour chaque problème, nous avons lancé le planificateur dix fois, cela comprend la génération de configurations finales, la planification aléatoire, l'optimisation de posture et l'optimisation de chemin. Nous indiquons le temps moyen pris par chacune de ces étapes, ainsi que les coûts relatifs de la détection de collision et de la résolution de cinématique inverse.

5.3.1 Scénario « Table » à double support

La figure 5.6 présente un chemin solution pour le problème consistant à atteindre un point sous une table avec la main droite. Une expérience identique a été présentée dans (Kanehiro *et al.*, 2008a). La méthode locale présentée dans (Kanehiro *et al.*, 2008a) calcule un pas d'optimisation en 100 ms pour ce problème. Sur le robot HRP-2, en choisissant le réglage qui consiste à donner des consignes de configuration toutes les 50 ms, cette méthode tourne à une demi-fois le temps réel. La trajectoire planifiée finalement dure 14 s, donc le temps de planification est de 28 s.

La table 5.1 présente les résultats de notre planificateur sur ce scénario. Le temps de calcul total, comprenant planification et optimisation, s'élève à 25,6 s. Notons que l'optimisation - de posture puis de trajectoire - est l'étape la plus consommatrice de temps de calcul. La planification proprement dite - génération de configurations finales puis de chemin - ne dure que 4,3 s. Toutefois, il est indispensable d'optimiser les résultats d'un algorithme de planification randomisés pour pouvoir utiliser le chemin résultat. La méthode locale présentée dans (Kanehiro *et al.*, 2008a) ne nécessite pas de post-optimisation, donc c'est bien 25,6 s qui doivent être comparées au 28 s de (Kanehiro *et al.*, 2008a).

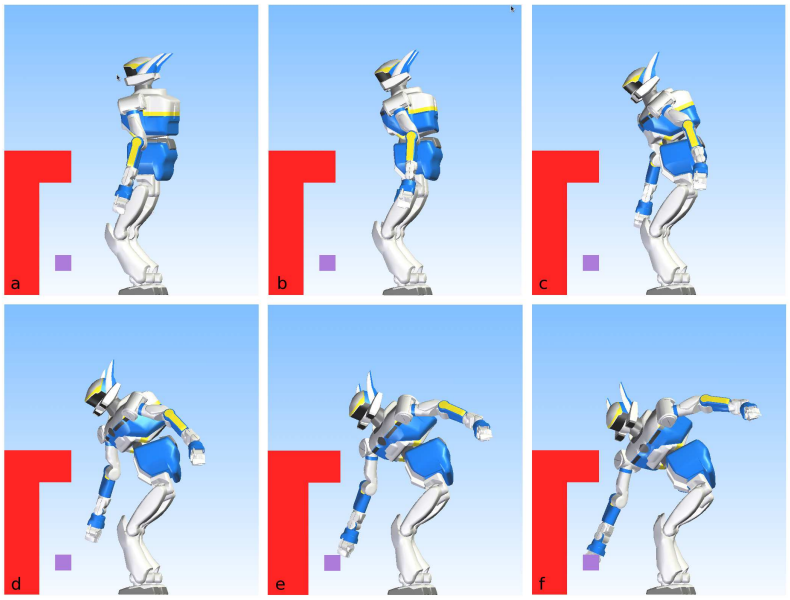


Figure 5.6 – Chemin solution pour une tâche d'atteinte de la main droite, tout en évitant l'obstacle que constitue la table rouge.

Génération de configurations finales	Planification	Optimisation de posture	Optimisation de chemin
281	4 017	16 181	5 096

Coût de la cinématique inverse	Coût du détecteur de collision
1,25%	86 %

Table 5.1 – Temps de calcul du planificateur sur le scénario « Table » en ms.

5.3.2 Scénario « Tore » à simple support

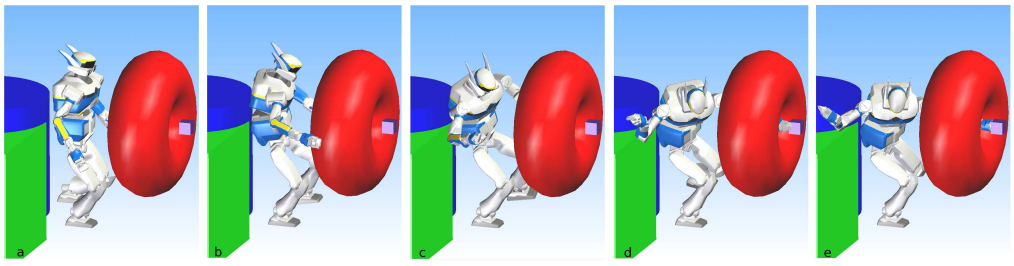


Figure 5.7 – Chemin solution pour une tâche d'atteinte de la main droite, tout en évitant les obstacles, et particulièrement le tore dans lequel doit passer la main.

La figure 5.7 présente un chemin solution pour une tâche d'atteinte plus difficile. Le robot tient en équilibre sur son pied droit et doit atteindre avec sa main gauche un point

à l'intérieur d'un obstacle de forme torique. Ce problème a été présenté dans (Kanehiro *et al.*, 2008b). Il s'agit d'un problème très coûteux à résoudre pour une méthode locale d'évitement d'obstacle en raison de la proximité et de la complexité des obstacles.

Le calcul d'un mouvement solution avec les modèles précis du robot et de l'environnement prend 1,47 s par pas d'optimisation. La raison de ce coût est le nombre élevé ($\simeq$ 3000) de contraintes induites par l'évitement d'obstacle. Les auteurs de (Kanehiro *et al.*, 2008b) proposent de résoudre le même problème en utilisant des modèles simplifiés de la géométrie du robot et de son environnement, ce qui ramène le temps de calcul à 50 ms par pas d'optimisation (pour 375 contraintes). La trajectoire générée dure 77 s. Ainsi, le temps de planification est de 77 s pour les modèles simplifiés et 1540 s pour les modèles complets.

Génération de configurations finales	Planification	Optimisation de posture	Optimisation de chemin
1 689	27 965	40 479	23 851

Coût de la cinématique inverse	Coût du détecteur de collision
9,19%	84 %

Table 5.2 – Temps de calcul du planificateur sur le scénario « Tore » en ms.

La table 5.2 présente les temps de calcul de notre planificateur sur ce scénario. Le temps de calcul total moyen est 94,0 s et le temps de planification 29,7 s. Encore une fois, l'essentiel du temps de calcul est utilisé pour l'optimisation de la posture finale et du chemin solution. La résolution de cinématique inverse coûte comparativement plus pour ce scénario, sans doute parce que la contrainte d'équilibre statique est plus difficile à résoudre avec un seul pied de support.

5.3.3 Comment prendre en compte l'évitement d'obstacle ?

Il est difficile de pousser plus loin la comparaison entre une méthode locale d'évitement d'obstacles et notre planificateur car ils ne sont pas prévus pour la même utilisation. La méthode locale peut être utilisée comme contrôleur tandis que le planificateur n'est utile que pour calculer des trajectoires hors ligne.

Toutefois, les deux exemples présentés ici peuvent donner une intuition de l'utilité de chacune des méthodes. L'intégration des contraintes d'évitement d'obstacles à la cinématique inverse permet de générer des mouvements plus réalistes que les algorithmes randomisés, qui génèrent des mouvements inutiles à la réalisation de la tâche but. Pour la résolution de problèmes simples, c'est-à-dire, où les calculs de collision ne sont pas coûteux, il est préférable d'utiliser de telles méthodes. En revanche, quand l'environnement est complexe, le paradigme de la planification de mouvement randomisée - qui utilise le détecteur de collisions comme un boîte noire plutôt que d'essayer de représenter explicitement $\mathcal{CS}_{\text{free}}$ - est pertinent et permet des résolutions beaucoup plus rapides.

5.4 Résultats expérimentaux

La section précédente s'est attachée à présenter une comparaison des performances de notre planificateur avec une méthode locale d'évitement d'obstacles. Nous présentons ici les résultats de notre planificateur sur différents problèmes de manipulation, pour lesquels les contraintes ne sont pas toujours les mêmes.

5.4.1 Mouvements d'atteinte

La tâche à résoudre ici est l'atteinte d'un point donné par la main droite du robot.

Atteinte d'objets sur une étagère

Dans ce premier exemple, nous avons placé l'objet à atteindre sur des étagères, à différentes hauteurs allant de 27 cm à 147 cm. Ce sont approximativement les limites de l'espace accessible pour le robot HRP-2. La figure 5.8 présente l'environnement et des postures solutions.

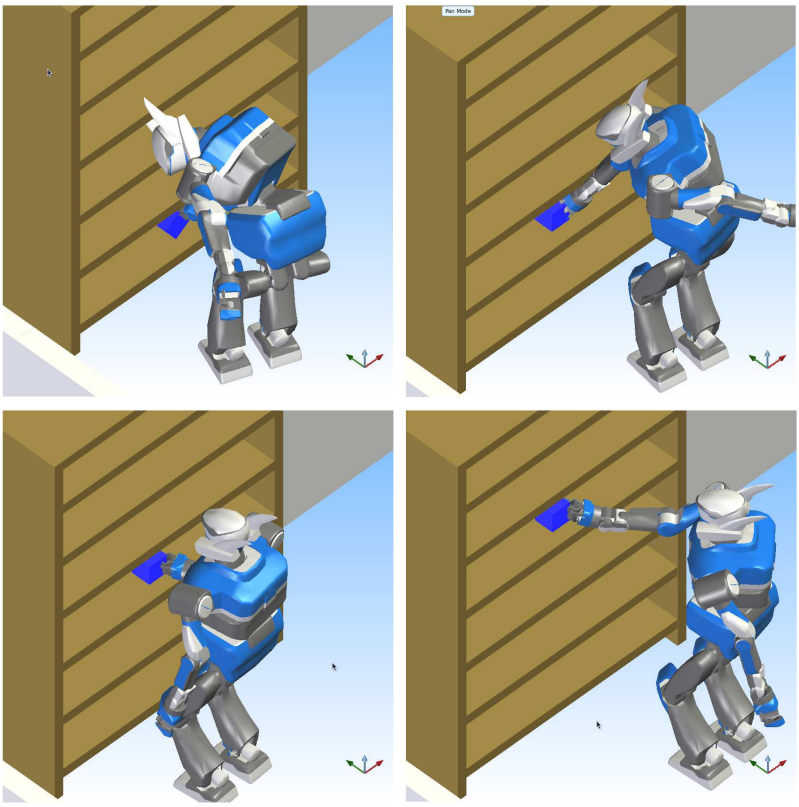


Figure 5.8 – Atteintes d'objets sur des étagères à différentes hauteurs.

La table 5.3 présente les résultats moyens de notre planificateur sur ce problème.

Hauteur		Moyenne	Min	Max
27 cm	Planification	34 486	8 037	89 711
	Optimisation	129 221	47 050	196 376
57 cm	Planification	41 999	5 800	155 887
	Optimisation	82 226	33 743	154 072
87 cm	Planification	27 677	9 638	85 286
	Optimisation	29 208	11 979	64 375
117 cm	Planification	41 878	9 809	103 139
	Optimisation	54 405	15 967	91 405
147 cm	Planification	52 911	16 389	119 458
	Optimisation	75 530	24 242	146 713

Table 5.3 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'atteinte d'un objet sur une étagère.

Atteinte d'objets dans un tiroir

La tâche d'atteinte présentée ici est plus complexe. Il s'agit d'atteindre un objet dans un tiroir, à deux hauteurs différentes : 56 cm et 91 cm. L'objet est placé 10 cm à l'intérieur du tiroir. Étant donnée la chaîne cinématique du robot HRP-2, très peu de configurations résolvent cette tâche sans entrer en collision avec le tiroir lui-même. Le problème est présenté en figure 5.9, et les résultats expérimentaux moyens de notre planificateur sont présentés en table 5.4.

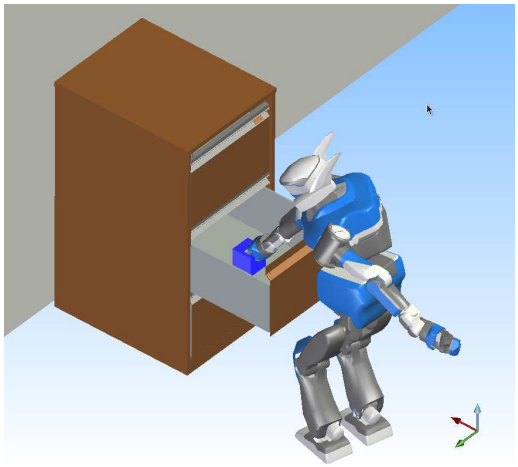


Figure 5.9 – Atteintes d'objets dans un tiroir à différentes hauteurs.

5.4.2 Manipulation d'objets en translation

La tâche consiste ici à attraper la poignée de l'objet à manipuler et à déplacer l'objet de 50 cm. Pendant la manipulation de l'objet en translation, une contrainte pour

Hauteur		Moyenne	Min	Max
56 cm	Planification	147 526	35 574	472 305
	Optimisation	151 734	83 671	216 723
91 cm	Planification	160 084	22 048	603 694
	Optimisation	208 544	114 579	284 482

Table 5.4 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'atteinte d'un objet dans un tiroir.

que la main du robot reste sur une droite donnée de l'espace de travail est ajoutée aux contraintes d'équilibre.

Pour ces problèmes, on ne définit pas de configuration finale, mais la condition implicite pour que le problème soit résolu est que l'objet se déplace d'au moins 50 cm. L'absence de configuration finale empêche de faire croître des arbres aléatoires enracinés en le but, ce qui ralentit la planification. Les temps présentés comprennent la planification du mouvement d'atteinte jusqu'à la poignée de l'objet à manipuler, ainsi que la manipulation proprement dite.

Ouverture d'une fenêtre coulissante

Le problème est présenté en figure 5.10 et les résultats moyens de notre planificateur en table 5.5. La poignée de la fenêtre à ouvrir est située à 140cm au dessus du sol.

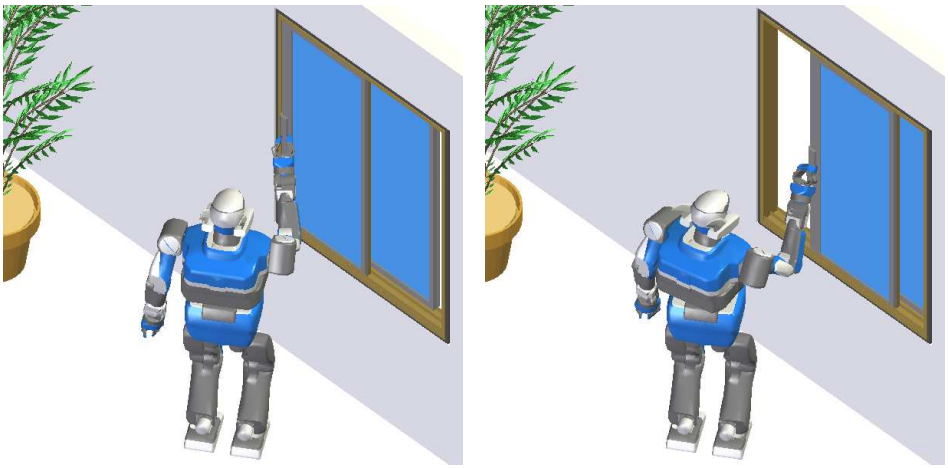


Figure 5.10 – Ouverture d'une fenêtre coulissante.

Ouverture d'un tiroir

Le problème est présenté en figure 5.11 et les résultats moyens de notre planificateur en table 5.6. La poignée du tiroir est située à 94 cm au dessus du sol.

Hauteur		Moyenne	Min	Max
140 cm	Planification	36 458	9 477	98 604
	Optimisation	23 341	9 474	43 381

Table 5.5 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'ouverture de fenêtre coulissante.

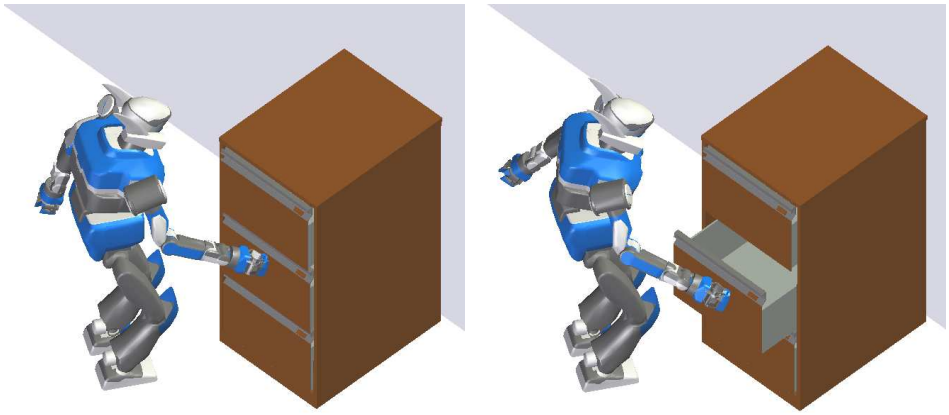


Figure 5.11 – Ouverture d'un tiroir.

Hauteur		Moyenne	Min	Max
94 cm	Planification	164 258	34 778	344 251
	Optimisation	185 850	64 525	271 092

Table 5.6 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'ouverture de tiroir.

5.4.3 Manipulation d'objets en rotation

Comme pour les objets en translation, le succès de la planification est défini implicitement, il s'agit d'ouvrir les objets à manipuler d'au moins 45°. Ici aussi, le mouvement à planifier comprend le geste d'atteinte jusqu'à la poignée de l'objet et le mouvement d'ouverture. Ces deux chemins sont ensuite optimisés.

Ouverture de porte

Le problème est présenté en figure 5.12 et les résultats moyens de notre planificateur en table 5.7. La poignée de la porte est située à une hauteur de 100 cm.

Ouverture d'une fenêtre

Le problème est présenté en figure 5.13 et les résultats moyens de notre planificateur en table 5.8. La poignée est située à une hauteur de 140 cm, un obstacle a été placé sous

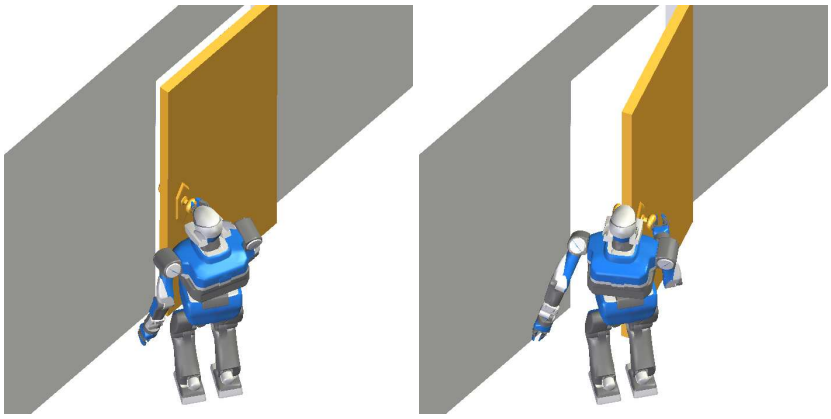


Figure 5.12 – Ouverture d'une porte.

Hauteur		Moyenne	Min	Max
100 cm	Planification	157 653	33 949	629 271
	Optimisation	22 163	10 637	46 158

Table 5.7 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'ouverture de porte.

la fenêtre, sa profondeur est de 20 cm et sa hauteur de 110 cm.

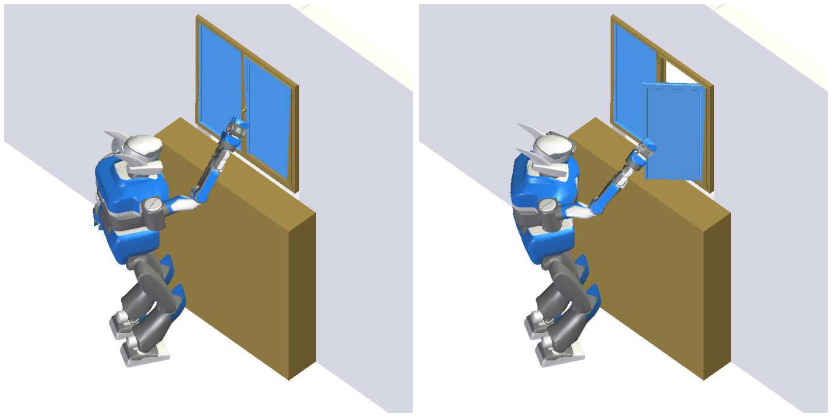


Figure 5.13 – Ouverture d'une fenêtre.

5.5 Conclusion

Nous avons présenté dans ce chapitre une méthode efficace pour utiliser les planificateurs de mouvement randomisés dans des problèmes sous contraintes. La génération de configurations respectant les contraintes est réalisée à l'aide de méthodes de cinématique inverse avec priorités. Nous avons comparé notre approche avec une technique

Hauteur		Moyenne	Min	Max
140 cm	Planification	127 059	29 805	343 035
	Optimisation	60 361	34 740	99 222

Table 5.8 – Temps de calcul (ms) pour la planification et l'optimisation d'un mouvement d'ouverture de porte.

consistant à intégrer l'évitement d'obstacle à la résolution des tâches de cinématique inverse. L'utilisation des planificateurs de mouvement randomisés permet de résoudre plus efficacement les problèmes à géométrie complexe.

Notre approche ressemble à celles présentées dans (Stilman, 2007) et (Berenson *et al.*, 2009). Toutefois, le formalisme de cinématique inverse avec priorités que nous utilisons est plus général, ce qui nous permet de considérer et résoudre une plus grande variété de tâches.

Chapitre 6

Planification de marche et manipulation

Le chapitre précédent a présenté un algorithme de planification corps-complet pour les problèmes où le polygone support du robot humanoïde est fixe. Au début du mouvement, le robot se tient sur ses deux pieds, ou sur un seul, et ses points de contact avec le sol ne changent pas durant le mouvement planifié.

Cette façon de procéder ne rend évidemment pas compte de toutes les facultés d'un robot humanoïde, qui peut se déplacer, en marchant, pour réaliser les tâches de manipulation souhaitées. La littérature présente de nombreuses méthodes pour planifier des mouvements de marche pour robot humanoïde. Le problème est souvent résolu sans prendre en compte de façon exacte l'évitement d'obstacles par ce qu'on appelle des générateurs de cycles de marche ou *walk pattern generator*. L'entrée de tels algorithme peut être des empreintes de pas au sol, ou une trajectoire à suivre pour le centre de masse du robot et leur sortie est une trajectoire articulaire corps-complet qui respecte des contraintes d'équilibre dynamique. Ces algorithmes supposent généralement que le robot commence et termine son mouvement de locomotion en une posture de référence qui correspond à la station debout chez l'humain. Pour le robot HRP-2, on parle de posture *demi-assise*, en effet, on choisit de garder les jambes du robot légèrement fléchies dans cette configuration de référence pour éviter les singularités apparaissant sur des tâches liées à la marche quand les jambes sont tendues.

Nous allons présenter brièvement dans une première section l'utilisation découplée de ces algorithmes avec le planificateur présenté dans le chapitre précédent. Dans un deuxième temps, nous présenterons une méthode plus puissante de planification de mouvements de manipulation nécessitant de marcher pendant la résolution de la tâche.

6.1 Découplage de la locomotion et de la manipulation

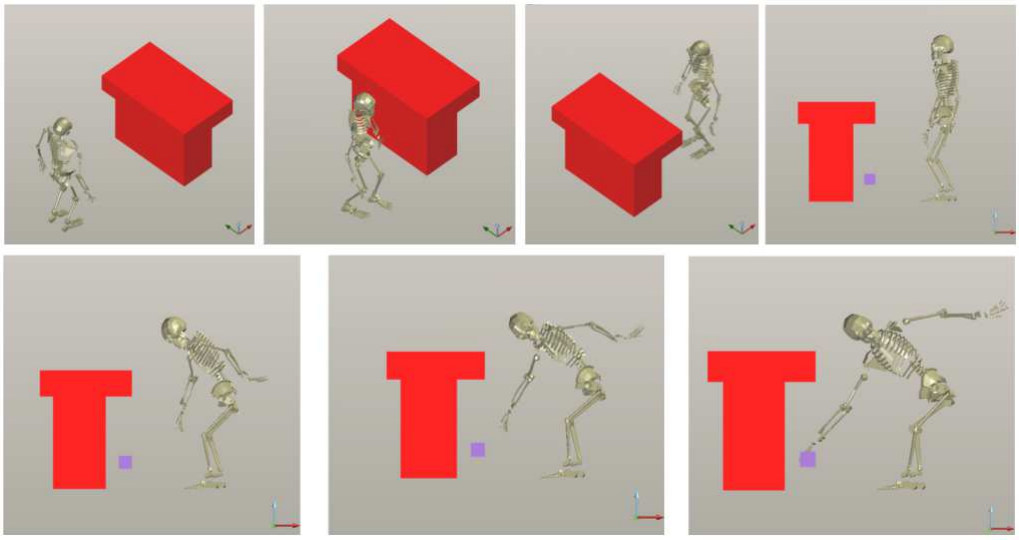


Figure 6.1 – Mouvement de marche puis corps-complet pour atteindre un objet sous une table.

Étant muni :
- d'un planificateur pour la navigation du robot - c'est-à-dire planifiant une trajectoire sans collision pour une boîte englobant le robot et ses mouvements de balancement générés dûs à la marche,
- d'un pattern generator,
- du planificateur corps-complet présenté dans le chapitre précédent,

on peut proposer une architecture simple pour résoudre nombre de problèmes usuels nécessitant marche puis mouvement corps-complet :

1. On échantillonne des configurations à proximité de l'objet à manipuler comme décrit dans le chapitre précédent, jusqu'à trouver au moins une configuration finale q_{final} résolvant la tâche de manipulation.
2. On génère la configuration du robot telle que le robot est dans sa posture de référence demi-assise et son polygone support est celui de la configuration finale qui résout la tâche : $q_{\text{final}}^{(\text{half-sitting})}$.
3. On résout le problème de navigation qui consiste à aller de la configuration initiale à $q_{\text{final}}^{(\text{half-sitting})}$.
4. On anime le chemin solution à l'aide du pattern generator.
5. On résout le problème de planification corps-complet qui consiste à aller de $q_{\text{final}}^{(\text{half-sitting})}$ à q_{final}.
6. On concatène les chemins solutions de (4) et (5).

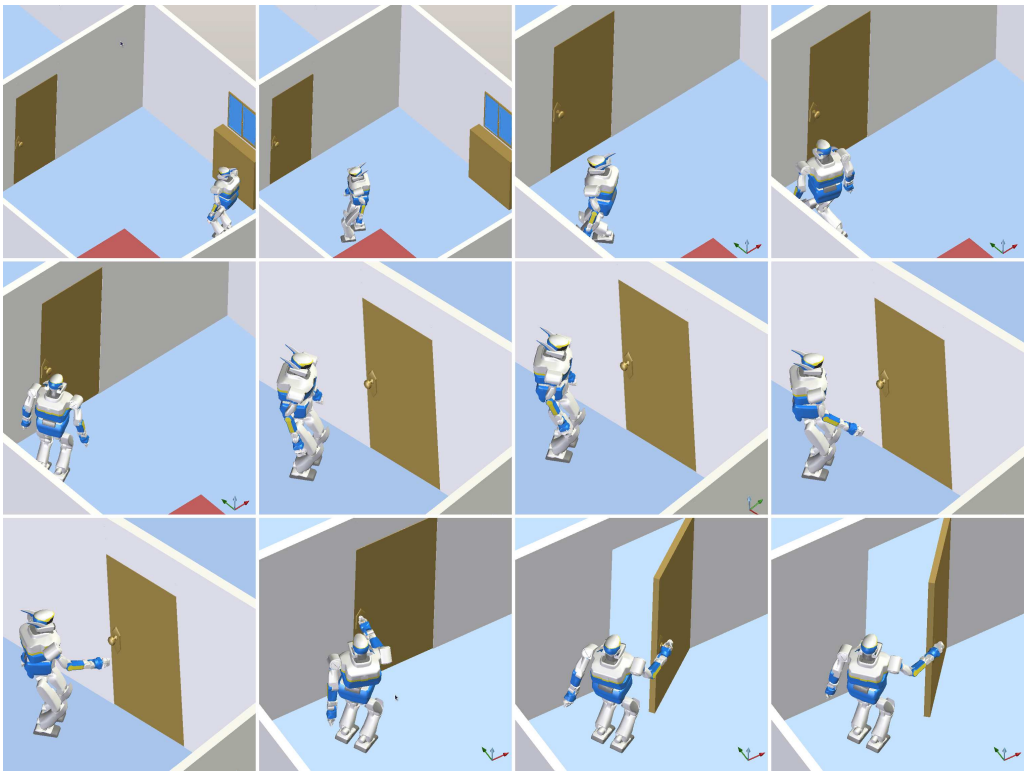


Figure 6.2 – Mouvement de marche puis corps-complet pour ouvrir une porte.

Les figures 6.1 et 6.2 présentent des exemples de chemins solutions générés par cette architecture pour des tâches d'atteinte d'un objet sous une table et d'ouverture de porte. Le planificateur de locomotion utilisé pour ces exemples est décrit dans (Yoshida *et al.*, 2007) et a été développé antérieurement au travail présenté ici.

6.2 Transformation d'un chemin corps complet en une trajectoire de marche

Le découplage présenté dans la section précédente, s'il est simple à mettre en œuvre, ne permet pas de résoudre les problèmes où le mouvement de marche doit s'accompagner de mouvements du haut du corps. Dans l'environnement présenté en figure 6.3, par exemple, le robot est obligé de lever les bras pour pouvoir passer entre les deux chaises en marchant. Une technique de planification de trajectoire de marche qui utilise une boîte englobante comme modèle du robot ne peut pas trouver de solution.
Nous proposons dans ce chapitre un planificateur corps-complet qui réalise des trajectoires de marche. Notons qu'on se restreint pour la génération de la marche aux mouvements qui satisfont le modèle simplifié de « chariot sur une table » présenté en section 3.2.2, c'est-à-dire qu'on ne peut générer que des trajectoires où le centre de masse du

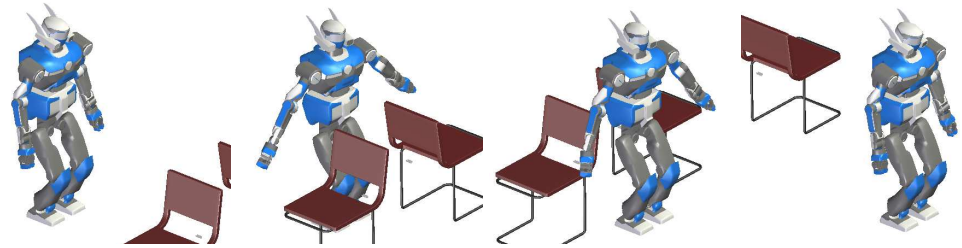


Figure 6.3 – Chemin sans collision pour un robot humanoïde glissant sur le sol. Les contraintes appliquées le long du chemin concernent : (i) les positions relatives des pieds, (ii) la position du centre de masse, (iii) la verticalité du bassin.

robot reste à hauteur constante.
L'extension des techniques du chapitre précédent à la génération de trajectoire corps-complet incluant la marche peut se faire de différentes façons. On peut par exemple échantillonner les positions des empreintes de pas, comme présenté dans (Perrin *et al.*, 2011), chaque pas correspond à une extension d'un algorithme de type RRT*, et des détections de collision valident le mouvement des jambes au cours de ces pas. Cette technique ne permet pas de planifier un mouvement de marche dans lequel le haut du corps nécessite aussi de la planification pour l'évitement d'obstacle.
Une autre stratégie, présentée dans (Pettré *et al.*, 2003) pour des acteurs digitaux et utilisée dans (Yoshida *et al.*, 2005) sur le robot humanoïde HRP-2 commence par planifier un mouvement pour un cylindre qui englobe les jambes du personnage puis à déformer la trajectoire des articulations du haut du corps localement si des collisions sont présentes lors de l'animation corps-complet de la trajectoire de marche. La limite de cette approche est qu'à la suite de la planification, il n'existe pas de garantie formelle que le chemin pourra être déformé en une trajectoire admissible par le système.

Dans ce chapitre, nous présentons une approche inspirée de (Pettré *et al.*, 2003), en deux temps. Dans un premier temps, on planifie le chemin corps-complet, sans collision, d'un robot glissant sur le sol, dont les deux pieds ont toujours la même position relative. Ce chemin est ensuite approximé avec une précision arbitraire par un contrôleur de marche dynamique qui utilise le modèle du chariot sur une table. La contribution de cette approche est la garantie formelle qu'à la suite de la planification pour le robot glissant, on pourra trouver une trajectoire de marche sans collision et à l'équilibre dynamique.

6.2.1 Trajectoire à l'équilibre statique

À l'aide des outils présentés dans le chapitre précédent, on planifie une trajectoire corps-complet pour un robot qui glisse sur le sol. Toutes les configurations échantillonnées vérifient des contraintes d'équilibre statique :
– les deux pieds sont à plat au sol,
– le centre de masse se projette verticalement au centre du polygone support.

On requiert de plus que le robot humanoïde respecte des contraintes qui permettent d'utiliser le modèle simplifié de chariot sur une table :
– le centre de masse reste à une hauteur constante,
– le bassin du robot reste vertical.
Notons que les valeurs de toutes ces contraintes peuvent être exprimées dans le référentiel du robot, donc les degrés de liberté non-articulaires du robot sont susceptibles de changer le long du chemin planifié.
La figure 6.3 présente un exemple de trajectoire pour le robot HRP-2 glissant sur le sol tout en respectant ces contraintes. L'évitement d'obstacle est obtenu par l'utilisation d'un algorithme RRT.

6.2.2 Existence d'une trajectoire de marche dynamiquement stable

La démonstration qu'une trajectoire de robot glissant obtenue dans la section précédente peut être approximée par une trajectoire de marche à l'équilibre dynamique utilise les raisonnements présentés en section 2.5 sur la contrôlabilité en temps petit.

Notations : Soit $\mathscr{C}$ l'ensemble des contraintes :

1. Le pied gauche du robot est au sol, et sa position relativement au bassin du robot est identique à celle en position demi-assise.
2. Le pied droit du robot est au sol, et sa position relativement au bassin du robot est identique à celle en position demi-assise.
3. Le centre de masse du robot se projette au centre du polygone support.
4. Le centre de masse du robot est à une hauteur z_c.
5. Le bassin du robot a une orientation verticale.

Soit $\mathscr{M}$ la sous-variété de $\mathcal{CS}$ constituée des configurations respectant $\mathscr{C}$.

Théorème 6.2.1. *Soit $q \in \mathscr{M}$ non singulière. Soit V un voisinage de q dans $\mathcal{CS}$. L'ensemble des configurations accessibles depuis q par une trajectoire à l'équilibre dynamique en restant dans V forme un voisinage de q dans $\mathscr{M}$.*

Ce résultat a pour conséquence la possibilité d'appliquer la méthode présentée en section 2.5. Après avoir planifié un chemin sans collision pour le robot glissant, on pourra l'approximer par une trajectoire de marche dynamique sans collision elle aussi. La section suivante détaillera l'algorithme effectif de génération de la trajectoire de marche.

Démonstration :

Ce théorème est valide dans le cadre des hypothèses utilisées dans le modèle de chariot sur une table, c'est-à-dire qu'on considère un robot de masse ponctuelle, se déplaçant à hauteur constante. Les bras sont considérés comme ayant une masse négligeable donc n'influent pas sur l'équilibre du robot, on peut donc utiliser une méthode locale linéaire

pour les degrés de liberté des bras. En revanche, les degrés de liberté des jambes ainsi que ceux qui positionnent le bassin dans l'espace (position assimilée à celle du centre de masse du robot) ne sont pas contrôlables par une interpolation linéaire. Ils doivent suivre un mouvement de marche à l'équilibre dynamique.
La démonstration du théorème 6.2.1 se fait en plusieurs étapes. Commençons par montrer qu'on peut suivre un chemin du robot glissant par une trajectoire de marche à l'équilibre dynamique en écartant arbitrairement peu le centre de masse de sa trajectoire initiale. Nous repérerons le robot dans le repère du monde à l'aide de la position de son centre de masse. En suivant le modèle de chariot sur une table, on requiert que la hauteur du centre de masse reste constante, ainsi que les rotations globales du robot autour des axes (x) et (y). Finalement, trois degrés de liberté non-articulaires servent à définir la position et l'orientation du robot dans le repère du monde : (x, y, θ) où x et y définissent la position horizontale de son centre de masse et θ l'angle de la rotation du robot autour de l'axe (z).

Mouvement de marche sur place. Commençons par montrer qu'on peut exécuter un mouvement de marche sur place à l'équilibre dynamique. Les équations qui régissent le mouvement du ZMP dans le plan horizontal en fonction du mouvement du centre de masse ont été présentées en section 3.2.2. On les réécrit ici :

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x - \frac{z_c}{g}\ddot{x} \\ y - \frac{z_c}{g}\ddot{y} \end{pmatrix} \tag{6.1}$$

Où (p_x, p_y) est la position horizontale du ZMP et (x, y) celle du centre de masse. Dans la suite, on notera $\omega_0 = \sqrt{\frac{g}{z_c}}$.
Avant de pouvoir lever un pied sans tomber, on doit faire passer le ZMP sous l'autre pied. Si on considère que le robot est orienté vers l'axe x, seule la composante y du ZMP nous intéresse. On garde donc les composantes en x du centre de masse et du ZMP constantes à zéro.
On souhaite réaliser le mouvement en bougeant arbitrairement peu le centre de masse. Soit $\epsilon > 0$ fixé quelconque, on impose pour tout $t > 0$, $|y(t)| \leq \epsilon$. Notons L la distance entre la projection du centre du robot au sol et le centre d'un pied. On souhaite faire osciller $p_y(t)$ entre $-L$ et L. Dans toute cette démonstration, on fera l'hypothèse que les pieds du robot sont rectangulaires, orientés parallèlement au robot, de longueur l_1 et de largeur l_2. Si les pieds ne sont pas rectangulaires, la démonstration peut être adaptée en considérant un rectangle inclus dans la surface de contact entre un pied et le sol. La démonstration n'est pas valide si cette surface de contact est nulle, par exemple pour un contact ponctuel.
L'idée est d'utiliser la forme de 6.1, pour appliquer un facteur d'échelle entre les oscillations du centre de masse et celles du ZMP. Par exemple pour $\omega \in \mathbb{R}_+^*$, si on applique $y(t) = \epsilon \sin(\omega t)$, on obtient $p_y(t) = (1 + \left(\frac{\omega}{\omega_0}\right)^2)\epsilon \sin(\omega t)$. L'amplitude des oscillations

de y est multipliée par un facteur $(1 + \left(\frac{\omega}{\omega_0}\right)^2)$. Pour $\omega = \omega_0\sqrt{\frac{L}{\epsilon} - 1}$, on obtient bien des oscillations de p_y entre $-L$ et L.
Si on part d'une configuration statique à $(t = 0)$ et qu'on désire commencer un mouvement de marche sur place, on ne peut pas appliquer directement la consigne $y(t) = \epsilon \sin(\omega t)$ car elle génère une discontinuité de la vitesse du centre de masse à $(t = 0)$. Pour remédier à cette discontinuité, on passe par un régime transitoire entre $(t = 0)$ et $(t = T_1)$.
Soit $f : [0, T_1] \to [0, 1]$ de classe C^∞ croissante telle que $f(0) = 0$, $\dot{f}(0) = 0$, $f(T_1) = 1$, $\dot{f}(T_1) = 0$ et $\ddot{f}(T_1) = 0$. On peut construire explicitement f en utilisant un polynôme de degré 4. On exige de plus que pour tout $t \in [0, T_1]$, $|2\epsilon\dot{f}(t)\frac{\omega}{\omega_0^2}| \leq \frac{l_2}{4}$ et $|\epsilon\ddot{f}(t)/\omega_0^2| \leq \frac{l_2}{4}$. On peut garantir ces conditions en choisissant T_1 suffisamment grand. Considérons maintenant le mouvement de centre de masse suivant :

$$y(t) = \begin{cases} f(t)\epsilon \sin(\omega t) & \text{si } t \in [0, T_1] \\ \epsilon \sin(\omega t) & \text{si } t \geq T_1 \end{cases}$$

On peut vérifier que y est C^2 sur $\mathbb{R}_+$, de dérivée 0 en $t = 0$. Quand $t \geq T_1$, on est dans le cas du régime permanent présenté au dessus, donc p_y oscille entre $-L$ et L, ce qui permet la marche sur place. Il reste à vérifier que pour $t \in [0, T_1]$, $p_y(t)$ défini par les équations 6.1 reste bien à l'intérieur du polygone support. Le calcul des dérivées successives de $y(t)$ donne :

$$p_y(t) = f(t)\epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2) \sin(\omega t) + 2\epsilon\dot{f}(t)\frac{\omega}{\omega_0^2}\cos(\omega t) + \frac{\epsilon}{\omega_0^2}\ddot{f}(t)\sin(\omega t)$$

Pour tout $t \in [0, T_1]$, $f(t)\epsilon(1 + \frac{\omega}{\omega_0}^2) \sin(\omega t)$ est compris entre $-L$ et L. D'après les hypothèses sur les dérivées de f, on a donc $p_y(t)$ compris entre $-L - l_2/2$ et $L + l_2/2$, ce qui garantit que le ZMP reste dans le polygone support.
La figure 6.4 présente un exemple de mouvement du centre de masse selon l'axe y et le mouvement du ZMP correspondant.
On peut passer du régime permanent de marche sur place à une configuration statique en appliquant un régime transitoire symétrique destiné à atténuer graduellement l'amplitude des oscillations du centre de masse sans provoquer de discontinuité de vitesse de la commande.

Suivi d'une trajectoire (x, y, θ). Les trajectoires du paragraphe précédent peuvent être adaptées pour effectuer un mouvement de marche sur place en (x_0, y_0, θ_0). Nous entendons par là que le centre de masse du robot en configuration statique est en (x_0, y_0), et que le robot est tourné d'un angle θ_0. Ainsi le centre du pied gauche du robot est en $foot_l(x_0, y_0, \theta_0) = ((x_0 - L\sin(\theta_0), y_0 + L\cos(\theta_0))$ et le centre du pied droit est en $foot_r(x_0, y_0, \theta_0) = (x_0 + L\sin(\theta_0), y_0 - L\cos(\theta_0))$. Notons $c_0(x_0, y_0, \theta_0) : \mathbb{R}_+ \to \mathbb{R}^2$ le mouvement du centre de masse correspondant :

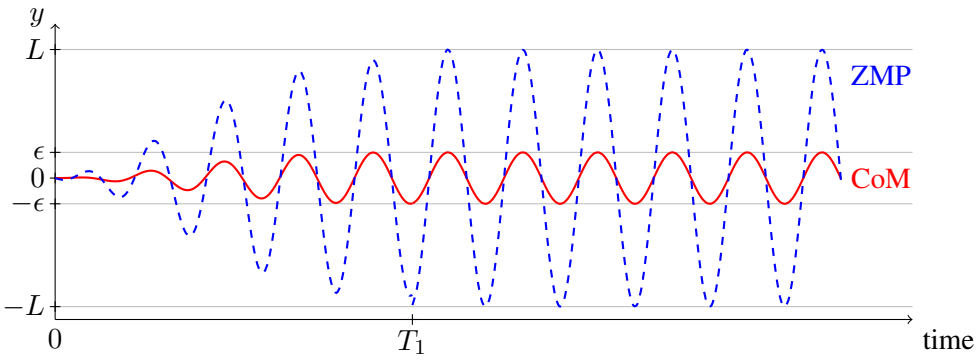


Figure 6.4 – Mouvement du centre de masse selon l'axe y. Le centre de masse reste dans l'intervalle $[-\epsilon, \epsilon]$ tandis qu'en régime permanent ($t \geq T_1$), le ZMP oscille entre les centres des deux pieds.

$$c_0(x_0, y_0, \theta_0)(t) = \begin{pmatrix} x_0 - \epsilon \sin(\theta_0) \sin(\omega t) \\ y_0 + \epsilon \cos(\theta_0) \sin(\omega t) \end{pmatrix}$$

Ce mouvement s'écarte de la position (x_0, y_0) de ϵ, et fait passer le ZMP successivement sous les centres des deux pieds. Notons zmp_0 la trajectoire de ZMP correspondante :

$$zmp_0(x_0, y_0, \theta_0)(t) = \begin{pmatrix} x_0 - \epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2) \sin(\theta_0) \sin(\omega t) \\ y_0 + \epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2) \cos(\theta_0) \sin(\omega t) \end{pmatrix}$$

Nous pouvons maintenant décrire le mouvement du centre de masse et des pieds du robot pour aller d'une configuration $q_i = (0, 0, 0)$ à $q_f = (x_f, y_f, \theta_f)$. On considère comme acquis le mécanisme de régime transitoire destiné à initialiser la marche sur place. À $(t = 0)$, le robot est en train de marcher sur place, son ZMP et son centre de masse sont en $(0, 0)$, en train de se déplacer vers le pied gauche.

Soit $f : [0, T] \to [0, 1]$ de classe C^∞, croissante telle que $f(0) = 0$, $\dot{f}(0) = 0$, $\ddot{f}(0) = 0$, $f(T) = 1$, $\dot{f}(T) = 0$ et $\ddot{f}(T) = 0$. On peut construire une telle f explicitement par un polynôme de degré 5. On impose de plus des limites sur les dérivées de f : pour tout $t \in [0, T]$,

$$\begin{aligned} \frac{1}{\omega_0^2}(|x_f| + |y_f|)|\ddot{f}(t)| &< \min(l_1/6, l_2/4) \\ \frac{2\epsilon\omega}{\omega_0^2}|\theta_f||\dot{f}(t)| &< l_1/6 \\ \frac{\epsilon}{\omega_0^2}|\theta_f||\ddot{f}(t)| &< l_1/6 \\ \frac{\epsilon}{\omega_0^2}\theta_f^2 \left(\dot{f}(t)\right)^2 &< l_2/4 \end{aligned}$$

Comme précédemment, on peut garantir ces bornes en choisissant T suffisamment grand, c'est-à-dire, quitte à ralentir l'exécution de la trajectoire.

La trajectoire $f_{CS}(t) = f(t)(q_f - q_i)$ va de q_i à q_f de 0 à T, sans sortir du segment $[q_i, q_f]$. Le mouvement du centre de masse destiné à suivre cette trajectoire en marchant s'écrit :

$$c(t) = c_0(f_{\mathcal{CS}}(t)) = \begin{pmatrix} f(t)x_f - \epsilon \sin(f(t)\theta_f)\sin(\omega t) \\ f(t)y_f + \epsilon \cos(f(t)\theta_f)\sin(\omega t) \end{pmatrix}$$

Notons que les conditions imposées aux bornes sur les dérivées première et deuxième de f garantissent la continuité de la consigne en vitesse du centre de masse et la continuité de la position du ZMP en 0.

La trajectoire qu'on souhaite obtenir pour le ZMP le long de la trajectoire est :

$$zmp_{des}(t) = zmp_0(f_{\mathcal{CS}}(t)) = \begin{pmatrix} f(t)x_f - \epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2)\sin(f(t)\theta_f)\sin(\omega t) \\ f(t)y_f + \epsilon(1 + \left(\frac{\omega}{\omega_0}\right)^2)\cos(f(t)\theta_f)\sin(\omega t) \end{pmatrix} \tag{6.2}$$

C'est celle qui fait passer le ZMP sous les centres des pieds en déplaçant les pieds comme suit : pour $n \in \mathbb{N}$, à $t_l^{(n)} = \frac{(2n+1/2)\pi}{\omega}$, le ZMP est sous le pied gauche, on déplace le pied droit en $foot_r(f_{\mathcal{CS}}(t + \pi/\omega))$, et en $t_r^{(n)} = \frac{(2n+3/2)\pi}{\omega}$, le ZMP est sous le pied droit, on déplace le pied gauche en $foot_l(f_{\mathcal{CS}}(t + \pi/\omega))$.
Notons $zmp_{real}(t)$ la trajectoire réelle du ZMP calculée à partir du mouvement du centre de masse $c(t)$. Le calcul des dérivées successives de $c(t)$ donne l'erreur entre $zmp_{des}(t)$ et $zmp_{real}(t)$:

$$\begin{aligned} zmp_{ref}(t) - zmp_{real} = \; & -\tfrac{1}{\omega_0^2}\ddot{f}(t)\begin{pmatrix} x_f \\ y_f \end{pmatrix} \\ & + \tfrac{2\epsilon\omega}{\omega_0^2}\theta_f \dot{f}(t)\cos(\omega t)\begin{pmatrix} \cos(f(t)\theta_f) \\ \sin(f(t)\theta_f) \end{pmatrix} \\ & + \tfrac{\epsilon}{\omega_0^2}\theta_f \ddot{f}(t)\sin(\omega t)\begin{pmatrix} \cos(f(t)\theta_f) \\ \sin(f(t)\theta_f) \end{pmatrix} \\ & - \tfrac{\epsilon}{\omega_0^2}\theta_f^2 \dot{f}(t)^2\sin(\omega t)\begin{pmatrix} \sin(f(t)\theta_f) \\ \cos(f(t)\theta_f) \end{pmatrix} \end{aligned} \tag{6.3}$$

Au temps t, le vecteur $\begin{pmatrix} \cos(f(t)\theta_f) \\ \sin(f(t)\theta_f) \end{pmatrix}$ est orienté dans la direction du robot. L'erreur suivant ce vecteur est inférieure à $l_1/2$. De même dans la direction orthogonale au robot, l'erreur est inférieure à $l_2/2$. Donc pour tout $n \in \mathbb{N}$, au temps $t_l^{(n)}$, le ZMP se projette bien sous le pied gauche et au temps $t_r^{(n)}$ sous le pied droit. Durant les phases de double support, le ZMP se projette entre les deux pieds. Toutes ces propriétés garantissent l'équilibre dynamique du mouvement de marche planifié, tout en gardant le centre de masse à une distance inférieure à ϵ du segment $[q_i, q_f]$.
Arrivé en q_f, une consigne continue du centre de masse permet de continuer en une marche sur place, puis le régime transitoire présenté précédement permet de revenir à une configuration statique.

Degrés de liberté des jambes. Le paragraphe précédent a montré qu'on pouvait réaliser un mouvement de marche en déplaçant arbitrairement peu le centre de masse du chemin pour le robot glissant. Il reste à quantifier le déplacement des degrés de liberté des jambes en fonction de la taille des pas et de la trajectoire du bassin souhaitée : Lors d'un mouvement de marche, les mouvements des jambes sont obtenus par la résolution de tâches de cinématique inverse qui définissent la position et l'orientation des pieds.
Pour une configuration du bassin donnée, la tâche qui s'applique aux degrés de liberté d'une jambe est définie par six équations qui fixent la position et l'orientation du pied. Notons q_{waist} la configuration du bassin (position et orientation), q_{foot} la configuration du pied désirée, et q_{leg} la configuration articulaire de la jambe considérée. q_{leg} est définie par six paramètres dans le cas qui nous intéresse (jambe d'un robot humanoïde). Notons Tk la fonction qui définit la tâche, $Tk : \mathbb{R}^6 \times \mathbb{R}^6 \times \mathbb{R}^6 \to \mathbb{R}^6$, $Tk(q_{waist}, q_{foot}, q_{leg}) = 0$. Tk s'écrit comme une somme de fonctions trigonométriques, et est donc de classe C^∞. Soit $q^{(0)} \in \mathcal{M}$. Par hypothèse, en $q^{(0)}$, les tâches définissant les positions et orientations des pieds ne sont pas en singularité, donc $\partial Tk/\partial q_{leg}(q^{(0)}_{waist}, q^{(0)}_{foot}, q^{(0)}_{leg})$ est inversible. Le théorème des fonctions implicites s'applique donc et garantit que :
il existe des ouverts $U \subset \mathbb{R}^{12}$ et $V \subset \mathbb{R}^6$ et $\phi : U \to V$ tels que :
– $(q^{(0)}_{waist}, q^{(0)}_{foot}) \in U$ et $q^{(0)}_{leg} \in V$,
– $\forall (q_{waist}, q_{foot}) \in U, Tk(q_{waist}, q_{foot}, \phi(q_{waist}, q_{foot})) = 0$,
– ϕ est de classe C^∞.
La continuité de ϕ implique que pour ϵ donné quelconque, il existe un ouvert U' contenant $(q^{(0)}_{waist}, q^{(0)}_{foot})$ et contenu dans U tel que pour tout $(q_{waist}, q_{foot}) \in U'$, $|\phi(q_{waist}, q_{foot}) - q^{(0)}_{leg}| < \epsilon$. Soit $U'_{waist} \subset \mathbb{R}^6$ et $U'_{foot} \subset \mathbb{R}^6$ des produits d'intervalles ouverts contenant respectivement $q^{(0)}_{waist}$ et $q^{(0)}_{foot}$, tels que $U'_{waist} \times U'_{foot} \subset U'$. Pour les configurations de $\mathcal{M}$, il existe une bijection $foot$ entre la configuration du bassin du robot et la configuration d'un des pieds. Soit U'' une boule ouverte contenant $q^{(0)}_{waist}$ inclue dans $U'_{waist} \cap foot^{-1}(U'_{foot})$. Soit h_{max} la hauteur maximale d'une configuration du pied dans U'_{foot}. Pour toute configuration $q \in \mathcal{M}$ telle que quand le robot est en q, son bassin est dans une configuration de U'', le suivi du chemin $[q^{(0)}, q]$ par des pas de hauteur inférieure à h_{max} générera des configurations de jambes à distance au plus ϵ de $q^{(0)}_{leg}$.

Conclusion de la preuve. Soit $q^{(0)}$ une configuration de $\mathcal{M}$ et $\epsilon > 0$ donné quelconque. Soit U''_l et U''_r des boules ouvertes de $\mathbb{R}^6$ obtenues par le paragraphe précédent pour les jambes gauche et droite respectivement. Soit V l'ensemble des configurations $q \in \mathcal{M}$ telles que :
– les degrés de liberté non-articulaires sont dans $U''_l \cap U''_r$,
– les degrés de liberté non-articulaires sont à distance au plus ϵ de ceux de $q^{(0)}$,
– les degrés de liberté du haut du corps en q sont à distance au plus ϵ de ceux en $q^{(0)}$.
V forme un voisinage de $q^{(0)}$ dans $\mathcal{M}$. Pour toute configuration $q \in V$, les mouvements du centre de masse et des jambes correspondant à une trajectoire de marche de $q^{(0)}$ à q comme décrits au-dessus, couplés à une interpolation linéaire des degrés de liberté du

haut du corps, génèrent une trajectoire de marche à distance au plus $k.\epsilon$ de $q^{(0)}$, où k est une constante dépendant du nombre de degrés de liberté du robot. Ceci conclut la preuve.

Remarque : Le contrôle présenté dans cette preuve peut générer des trajectoires très longues en temps, en raison des états transitoires au début et à la fin de la locomotion. Dans l'implémetation réelle, on a choisi de générer les mouvements du centre de masse par un contrôleur prédictif de ZMP, comme présenté dans (Kajita *et al.*, 2003). Nous avons observé expérimentalement que l'amplitude des oscillations du centre de masse décroissait quand la fréquence des pas augmentait.

6.2.3 Animation du chemin du « robot glissant »

La démonstration de la section précédente a inspiré notre algorithme pour animer le chemin du robot glissant en une trajectoire de marche à l'équilibre dynamique.
On commence par placer les empreintes de pas qui correspondent à la marche nominale du robot - ici HRP-2 - le long du chemin du robot glissant. L'animation se fait en utilisant le mécanisme présenté dans (Yoshida *et al.*, 2006) qui inclut le contrôle du ZMP du robot humanoïde dans la résolution de tâches de cinématique inverse hiérarchisée. À ce stade, les tâches appliquées au robot sont donc, par ordre de priorité décroissant :

1. Position et orientation du pied en vol,
2. Position horizontale du centre de masse,
3. Hauteur du bassin du robot,
4. Verticalité du bassin du robot,
5. Tâche de configuration vers le chemin du robot glissant.

Les tâches (1) et (2) génèrent une marche à l'équilibre dynamique, les tâches (3) et (4) assurent que le modèle simplifié de « chariot sur une table » est vérifié. La tâche (5), qui n'est pas nécessairement résolue, sert à approximer le chemin initialement planifié.
Parce que la tâche (5) est en plus faible priorité, elle n'est pas toujours résolue. Des collisions peuvent donc apparaître lors de l'animation du chemin du robot glissant en trajectoire de marche. Des collisions apparaissent si l'animation dynamique s'écarte trop du chemin pour le robot glissant. Si c'est le cas, il faut approximer plus finement le chemin initial, en utilisant la propriétés de contrôlabilité en temps petit énoncée dans la section précédente. La mise en œuvre de cette approximation suit celle présentée dans (Laumond *et al.*, 1994) pour les robots mobiles.

Si le chemin animé présente des collisions avec l'environnement, on coupe le chemin en deux sous-chemins, qu'on anime récursivement. Quand les chemins à animer sont trop courts pour les paramètres de marche nominale du robot, on accélère les pas. Comme

présenté dans la démonstration de la section précédente, la trajectoire de marche dynamique pour des pas de plus en plus courts et de plus en plus rapides approxime arbitrairement près le chemin du robot glissant.
L'algorithme 6 présente le pseudo-code correspondant au passage d'un chemin p pour le robot glissant à une trajectoire de marche dynamique.

Algorithme 6 FindDynamicTrajectory(Path p)

$Footprints \leftarrow$ ComputeFootprints(p)
$StackOfTasks$.initialize()
$StackOfTasks$.addFootprintTask($Footprints$)
$StackOfTasks$.addWaistTask()
$StackOfTasks$.addConfigurationTask(p)
$DynamicTrajectory \leftarrow$ Animate($StackOfTasks$)
if CheckForCollisions($DynamicTrajectory$) = Colliding **then**
  $(p_1, p_2) \leftarrow$ CutInHalf(p)
  $DynamicTrajectory_1 \leftarrow$ FindDynamicTrajectory(p_1)
  $DynamicTrajectory_2 \leftarrow$ FindDynamicTrajectory(p_2)
  return Concatenate($DynamicTrajectory_1, DynamicTrajectory_2$)
else
  return $DynamicTrajectory$
end if

La fonction `ComputeFootprints()` utilise la longueur du chemin p passé en argument pour déterminer s'il faut accélérer les pas, c'est-à-dire les oscillations du bassin du robot. La fonction `Animate()` est un appel aux algorithmes de cinématique inverse incluant la génération de trajectoires du centre de masse pour satisfaire les critères de stabilité dynamique liés au ZMP.

6.3 Résultats expérimentaux

Les résultats présenté ici ont été obtenus sur un PC Intel Core 2 Duo 2,13 GHz avec 2 Go de RAM. Pour chaque exemple, les temps de calculs des algorithmes de planification aléatoires sont la moyenne de cinquante exécutions de l'algorithme.

6.3.1 Passage entre deux chaises

L'environnement des figures 6.3 et 6.5 a été présenté dans (El Khoury *et al.*, 2011). Les auteurs ont résolu le problème de planification en utilisant une méthode de boîte englobante, ce qui contraint le robot à passer entre les chaises en marchant sur le côté. Notre planificateur est capable de résoudre ce problème en générant une marche en avant, ce qui peut être nécessaire si le robot doit utiliser la vision pendant la locomotion, par exemple. La première phase de planification consomme 29,6 s de temps de calcul en moyenne, et l'animation du chemin 66,5 s.

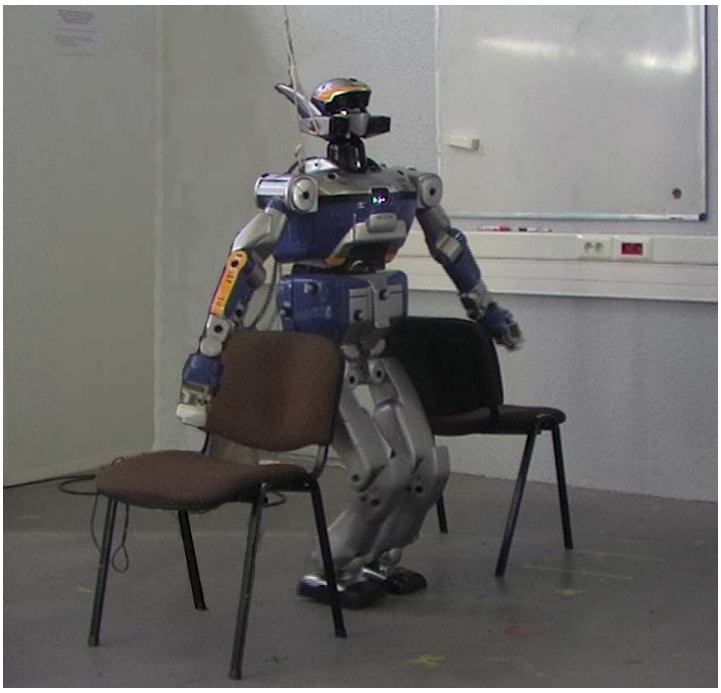


Figure 6.5 – Le robot HRP-2 qui passe entre deux chaises en marchant.

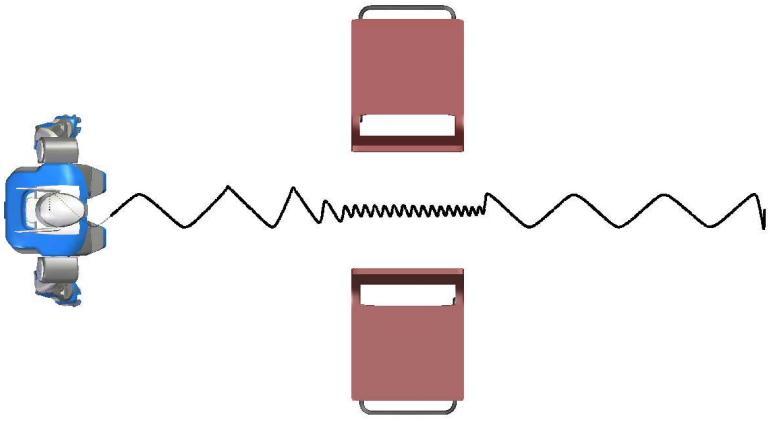


Figure 6.6 – Trajectoire horizontale du bassin du robot pendant la locomotion. Quand le robot est proche des obstacles, l'amplitude des oscillations diminue.

La figure 6.6 montre la trajectoire horizontale du bassin du robot pendant la locomotion. Cette trajectoire approxime celle du centre de masse. On peut observer la décroissance de l'amplitude des oscillations lors du passage entre les chaises. Ce mouvement a été validé sur une plateforme réelle. Pour fluidifier les mouvements du robot réel, une fois la suite de trajectoires de marche obtenue, on a concaténé toutes les empreintes de pas, ainsi que leurs paramètres temporels, pour générer une unique trajectoire de marche dynamique.

On a ensuite validé cette nouvelle trajectoire dans un détecteur de collisions.

6.3.2 Marche dans un environnement encombré

L'environnement présenté en figure 6.7 a été inspiré par les problèmes d'animation graphique. Le robot doit trouver un chemin sans collision au milieu de nombreux obstacles qui flottent dans les airs. La première étape de planification prend 184,3 s en moyenne, et l'animation du chemin en trajectoire dynamiquement stable 339,5 s. La figure 6.8 présente la trajectoire horizontale du bassin du robot pendant la locomotion.

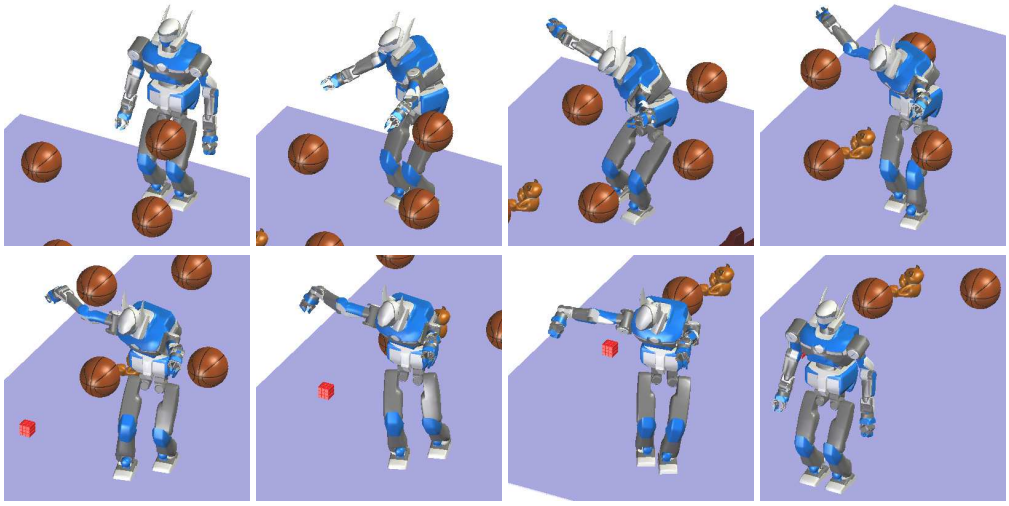


Figure 6.7 – Trajectoire de marche solution dans un environnement encombré. Le robot marche au milieu d'obstacles flottant dans les airs.

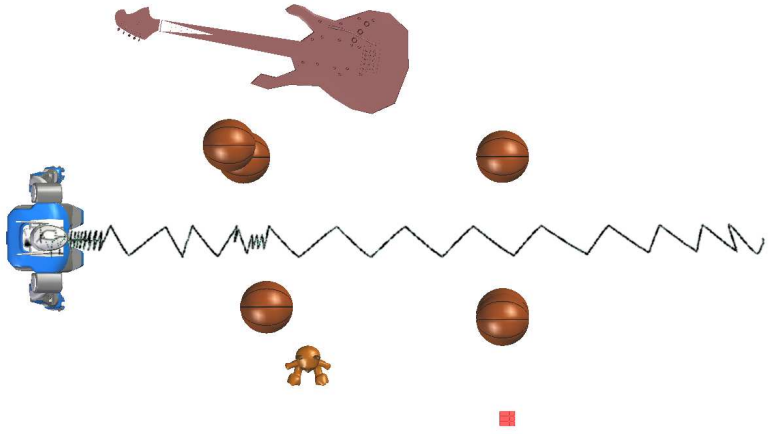


Figure 6.8 – Trajectoire horizontale du bassin du robot pendant la locomotion.

6.3.3 Planification d'un geste d'atteinte

Le problème présenté en figure 6.9 reprend le formalisme du chapitre précédent. Le but est défini implicitement par une tâche d'atteinte à réaliser par le robot. La configuration finale a été générée automatiquement, puis on a lancé le planificateur. La première étape de planification prend 83,0 s en moyenne, et l'animation du chemin 90,1 s. La figure 6.10 présente la trajectoire horizontale du bassin du robot pendant la locomotion.

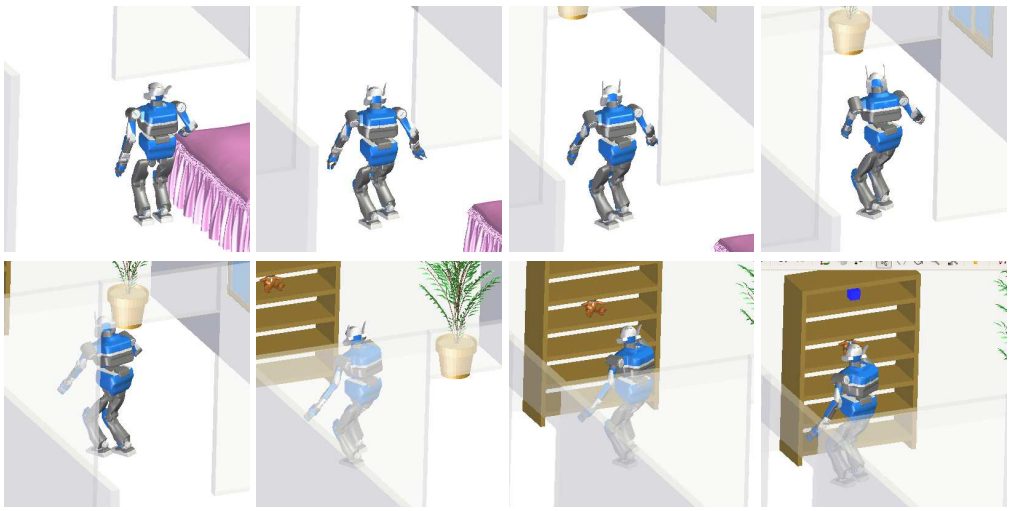


Figure 6.9 – Trajectoire solution pour un geste d'atteinte. Le but est défini implicitement par une tâche de cinématique inverse sur la main droite du robot.

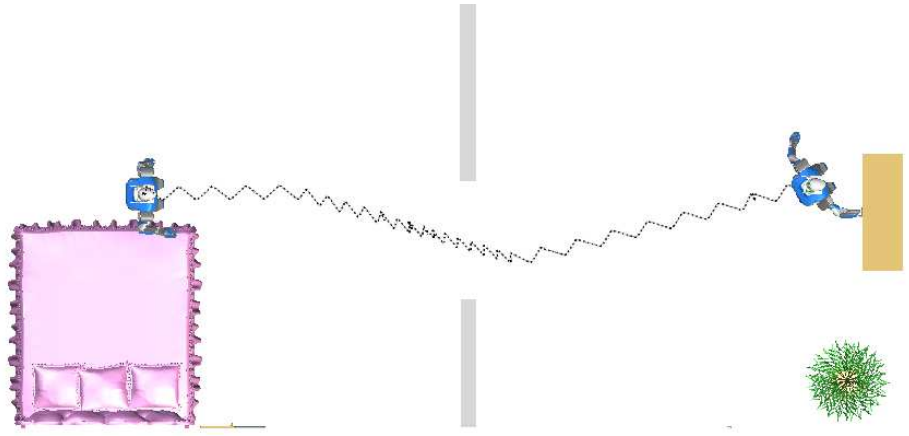


Figure 6.10 – Trajectoire horizontale du bassin du robot pendant la locomotion.

6.4 Conclusion et limites

6.4.1 Contribution

Dans ce chapitre, nous avons présenté une méthode originale pour généraliser les méthodes de planification corps-complet du chapitre précédent. Le planificateur présenté fonctionne en deux temps : une phase de planification puis une phase de placement des pas et d'animation dynamique. Le fait que la phase d'animation converge est garantie par une propriété de commandabilité en temps petit dont on a fait la démonstration sous les hypothèse du modèle « chariot sur une table ».

6.4.2 Limites

Une des limites de cette méthode est due à l'expression des contraintes pendant la phase de planification. Le fait qu'on projette les pieds au sol ne permet pas de planifier des mouvements de marche qui enjambent des obstacles. Il s'agit pourtant d'une caractéristique utile des robots humanoïdes, la poursuite de ce travail consistera en l'intégration de cette possibilité.
L'autre principale limite est dûe à l'utilisation du modèle simplifié du « chariot sur une table ». Le fait qu'on limite le robot à des mouvements pendant lesquels son centre de masse reste à hauteur constante ne traduit pas toute les possibilités d'un robot humanoïde, et ne permet pas de se baisser pour passer en marchant sous des obstacles, par exemple. (Kanehiro *et al.*, 2004) présente le couplage de plusieurs stratégies de marche, il pourrait être intéressant de les intégrer à notre planificateur.

Chapitre 7

Manipulation d'objets documentés

Les deux chapitres précédents ont présenté des algorithmes de planification corps complet pour résoudre des tâches simples, en utilisant la marche ou non. Par tâches simples nous entendons qu'elles s'expriment comme une pile de tâches exprimables dans le formalisme de la cinématique inverse présenté en section 3.1.
Certains problèmes de manipulation de la vie quotidienne ne s'inscrivent pas dans le formalisme de la section 3.1. Le passage d'une porte, par exemple, qui sera le problème canonique étudié dans ce chapitre, consiste en une suite de tâches élémentaires : (i) aller jusqu'à la porte, (ii) attraper la poignée, (iii) ouvrir la porte, etc. La planification globale du passage d'une porte ne peut cependant pas être résolue en mettant bout-à-bout les solutions à ces problèmes. En effet, la présence d'obstacles sur le chemin de la porte ou du robot - pendant la tâche (iii) - peut influencer la posture que doit adopter le robot au moment de la tâche d'atteinte de la poignée (ii).
Ce chapitre présente une contribution destinée à planifier des mouvements de manipulation complexes, par exemple le passage d'une porte, de façon efficace. Plutôt que de s'attaquer au problème de la manipulation sous sa forme générale, nous choisissons ici de présenter un modèle de représentation simplifiée de l'information propre aux objets manipulés. Nous appellerons cette information la « documentation d'un objet ».

7.1 Documentation d'un objet

La documentation d'un objet est la formalisation de l'information utilisable par un planificateur de mouvement pour simplifier un problème de manipulation à contraintes variables.
Formellement, nous représentons cette documentation comme un graphe, dont chaque nœud représente une contrainte, et les arêtes représentent la possibilité de passer d'une

contrainte à une autre. La figure 7.1 présente un exemple de documentation pour le passage d'une porte.

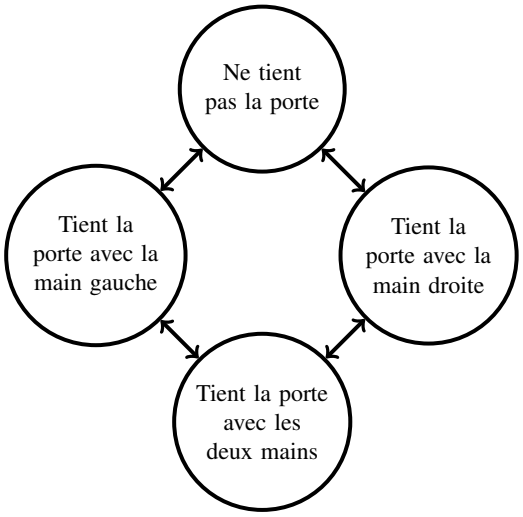


Figure 7.1 – Documentation d'une porte : graphe des contraintes appliquées au robot lors du passage d'une porte en utilisant ses deux mains.

À chacun des nœuds de ce graphe correspond une ou plusieurs contraintes ou tâches de cinématique inverse. La figure 7.2 présente des exemples de configurations qui résolvent chacune des tâches de la documentation d'une porte. Dans toutes ces configurations, les algorithmes de cinématique inverse garantissent, outre les tâches définies par la documentation de l'objet, des contraintes d'équilibre statique.

7.2 Adaptation des planificateurs randomisés

Un certain nombre d'adaptations sont nécessaires pour utiliser la documentation d'un objet en planification de mouvement. Nous les présentons dans cette section.

7.2.1 Représentation du système

On planifie le mouvement d'un système comprenant le robot humanoïde et un objet manipulé. L'état, ou la configuration du système est entièrement décrite par la donnée :
– des degrés de liberté du robot,
– des degrés de liberté de l'objet manipulé,
– d'un degré de liberté supplémentaire, à valeurs discrètes, qui indique quel type de contrainte s'applique au robot humanoïde.

Par exemple, pour la documentation d'une porte présentée en figure 7.1, ce dernier degré de liberté peut prendre quatre différentes valeurs, chacune correspondant à un nœud du graphe.

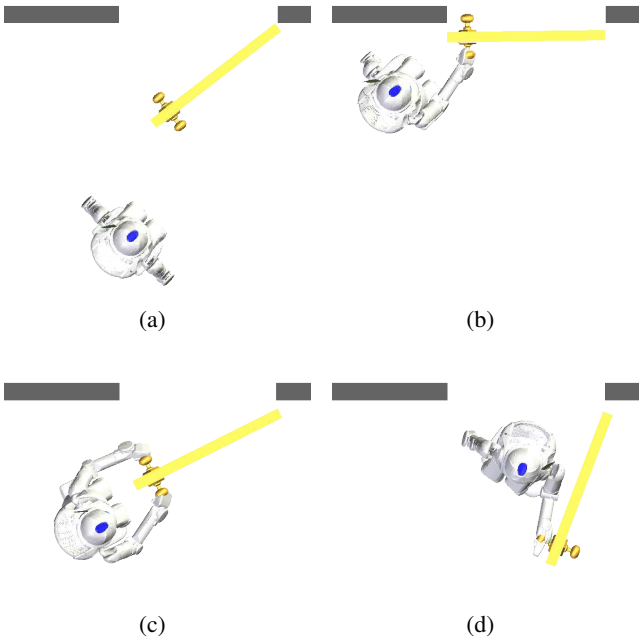


Figure 7.2 – Exemples de contraintes appliquées à un robot humanoïde lors du passage d'une porte. Les états du système (robot,porte) respectifs sont : (a) « Ne tient pas la porte » ; (b) « Tient la porte avec la main droite » ; (c) « Tient la porte avec les deux mains » ; (d) « Tient la porte avec la main gauche ».

Du point de vue de l'implémentation, dès qu'un objet configuration est créé ou modifié, le projecteur correspondant à la valeur de ce degré de liberté supplémentaire est appelé sur les degrés de liberté du robot humanoïde.

7.2.2 Interpolation entre deux configurations

Nous venons de présenter une représentation du système (robot,objet) qui comprend une information sur le lien entre le robot et l'objet en chaque configuration. Avec cette représentation, il existe des couples de configurations (q_1, q_2) tels qu'il n'existe pas de chemin élémentaire entre q_1 et q_2. Deux causes peuvent empêcher de créer un chemin élémentaire entre deux configurations :

1. Si la connexion entre deux configurations n'est pas autorisée par la documentation de l'objet. Par exemple, sur la figure 7.1, on ne peut pas passer directement d'une configuration où le robot tient la porte de la main droite à une configuration où il tient la porte de la main gauche.
2. L'objet manipulé ne peut pas se déplacer si le robot ne l'a pas en main. L'échantillonnage aléatoire de configurations du système va générer différentes valeurs pour les degrés de liberté de l'objet manipulé. Toutefois, une configuration dans

laquelle le robot ne tient pas l'objet ne peut pas être reliée à une autre configuration dans laquelle l'objet s'est déplacé.

Quand un chemin élémentaire n'existe pas entre deux configurations, la méthode locale ne renvoie rien. Cela ne change pas la structure des algorithmes de planification.
Quand il existe un chemin élémentaire entre deux configurations pour lesquelles le degré de liberté supplémentaire varie, la contrainte à appliquer le long du chemin est la plus faible des deux contraintes aux bouts du chemin. Par exemple, si on va d'une configuration q_1 dans laquelle le robot ne tient pas la porte à une configuration q_2 où il la tient dans la main droite, le chemin élémentaire entre q_1 et q_2 est constitué de configurations où le robot ne tient pas la porte.

7.2.3 Distance

Un algorithme RRT est sensible à la métrique utilisée dans $\mathcal{CS}$. L'opérateur *distance*, qui sert à choisir les configurations qu'on étend, doit refléter la probabilité que le chemin élémentaire entre deux configurations valides soit valide aussi. Jusqu'ici, cette validité ne concernait que la détection de collision. Comme notre représentation du système (robot,objet) n'autorise pas tous les chemins élémentaires possibles, l'opérateur distance qu'on utilise doit être modifié. Si, pour $(q_1, q_2) \in \mathcal{CS}^2$, il n'existe pas de chemin élémentaire entre q_1 et q_2, pour une des deux raisons évoquées dans le paragraphe précédent, on pose $d(q_1, q_2) = +\infty$.

7.2.4 Adaptation de l'échantillonnage

Comme dit précédemment, une contrainte sur le système (robot, objet) est formellement une sous-variété ou une famille de sous-variété de $\mathcal{CS}$. Passer d'une contrainte à une autre requiert d'échantillonner une configuration à l'intersection de telles sous-variétés.
Toujours en suivant l'exemple de l'ouverture d'une porte, pour passer d'une configuration q_1 dans laquelle le robot ne tient pas la porte à une configuration q_2 dans laquelle le robot tient la porte avec la main droite, le planificateur doit générer une configuration intermédiaire q_i dans laquelle le robot tient la porte, et la porte a la même configuration que dans q_1. La probabilité d'échantillonner une telle configuration est 0. Pour résoudre ce problème, nous utilisons un échantillonnage adapté.
Quand une configuration q est présente dans $\mathcal{R}$, on échantillonne spécifiquement les intersections de la sous-variété correspondant à la contrainte appliquée à q avec celles correspondant aux autres contraintes accessibles depuis q.

7.3 Représentation du robot humanoïde

Une simplification courante dans la littérature pour la planification de marche d'un robot humanoïde consiste à ne considérer qu'une boîte englobant le robot lors de la détection de collision. Au lieu de planifier pour tous les degrés de liberté en même temps, on

planifie le mouvement de cette boîte dans le plan, en se ramenant ainsi à la planification pour un système à trois degrés de liberté (x, y, θ) et en simplifiant les calculs du détecteur de collision.
Le formalisme des objets documentés se prête à cette simplification. Il faut cependant réécrire les contraintes de cinématique inverse pour pouvoir les appliquer à une boîte et non plus au robot humanoïde. Nous le faisons ici, en approximant l'espace accessible par les mains du robot par des sphères, et en projetant les degrés de liberté de la boîte pour que la poignée de la porte se trouve à l'intérieur d'une de ces sphères quand la contrainte l'impose.
La figure 7.3 reprend les exemples de contraintes de la figure 7.2 en les appliquant à un modèle de boîte englobante.

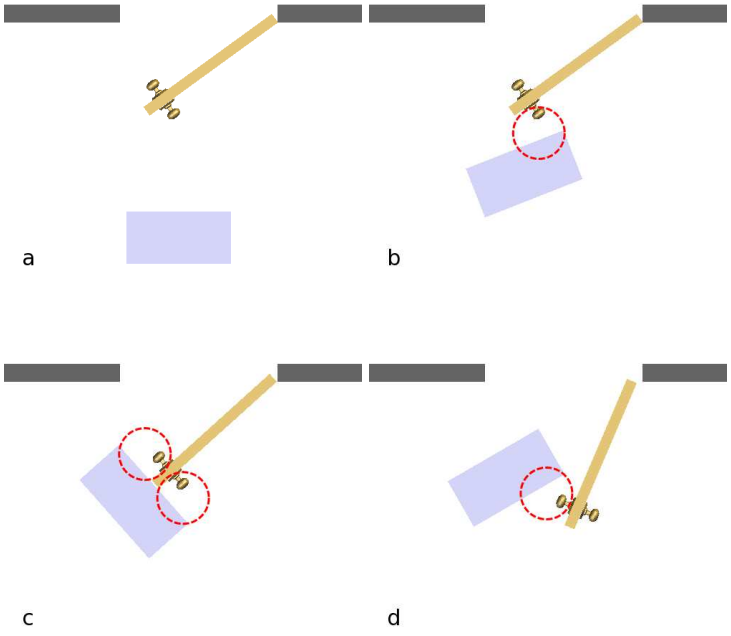


Figure 7.3 – Différentes contraintes de la documentation d'une porte appliquées à un modèle simplifié du robot humanoïde : une boîte englobant ses mouvements lorsqu'il marche. On approxime l'espace de travail de chacune des mains du robot par une sphère, représentée en pointillés rouges sur cette figure.

7.4 Animation du chemin résultat

Muni d'une documentation d'un objet, on peut planifier un chemin de passage d'une porte, soit par un modèle de boîte englobante, soit par un robot humanoïde qui glisse sur le sol. La figure 7.4 présente un exemple de chemin solution pour la boîte englobante.

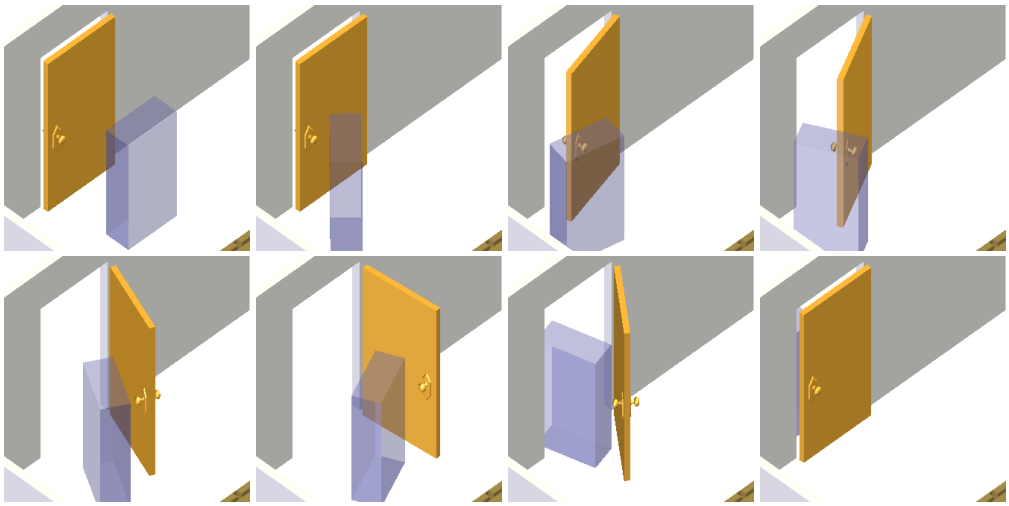


Figure 7.4 – Chemin solution pour le passage d'une porte par une boîte englobante.

Une fois qu'un tel chemin a été trouvé, il reste à l'animer par un mouvement de marche dynamique. Si on a planifié un chemin corps-complet pour le robot humanoïde qui glisse sur le sol, on peut l'animer avec l'algorithme présenté dans le chapitre 6. Si on a planifié un chemin pour la boîte englobante, on calcule les empreintes de pas correspondantes et on génère une trajectoire de marche à l'équilibre dynamique qui réalise les tâches de manipulation correspondant à la valeur du degré de liberté supplémentaire.
Les approximations et heuristiques que constituent la taille de la boîte englobante et les sphères qui approximent les espaces de travail du robot n'apportent pas de garantie formelle que le chemin de la boîte pourra être animé en une trajectoire corps-complet du robot humanoïde. En revanche, si on a planifié une trajectoire corps-complet de robot glissant, l'analyse du chapitre 6 garantit qu'on pourra l'animer en une trajectoire de marche à l'équilibre dynamique.

7.5 Résultats expérimentaux

Cette section présente des résultats expérimentaux de notre planificateur utilisé sur le robot humanoïde Partner de Toyota^{TM}et sur le robot HRP-2. La contribution de ce travail est la formalisation de la documentation des objets, nous ne nous sommes donc pas concentrés sur l'évaluation précise de cette méthode en termes de temps de calcul. Pour tous les exemples présentés dans cette section, la planification du mouvement de boîte englobante sur un PC Intel Core 2 Duo 2,13 GHz avec 2 Go de RAM a pris entre quelques dizaines de secondes et quelques minutes. Notons que l'animation du chemin obtenu utilise des méthodes de cinématique inverse qui tournent en temps réel sur le robot, donc le temps de calcul pour cette deuxième étape n'est pas limitant.

7.5.1 Passage d'une porte

La figure 7.5 présente l'animation du chemin pour la boîte englobante présenté en figure 7.4.

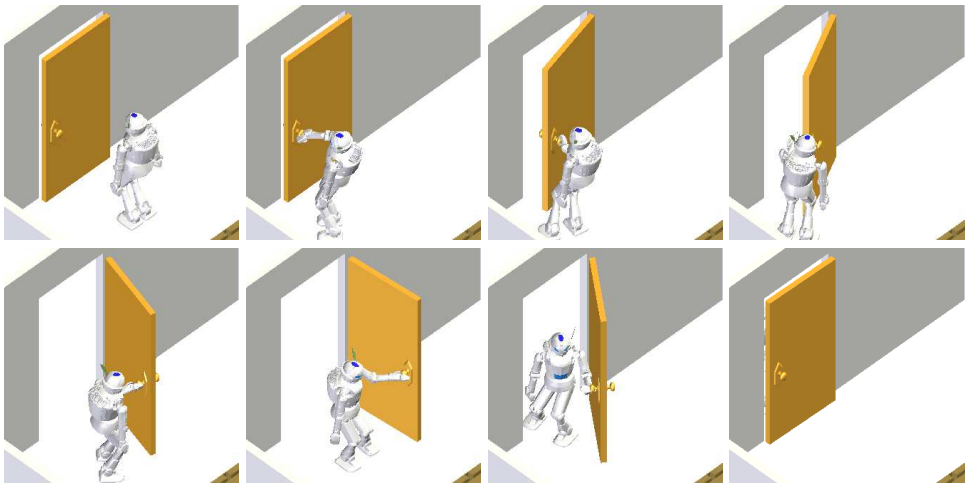


Figure 7.5 – Trajectoire de marche à l'équilibre dynamique pour le robot Partner qui passe une porte.

La figure 7.6 présente le même problème, dans lequel la planification est compliquée par la présence d'une chaise qui empêche l'ouverture de la porte en grand.

7.5.2 Passage d'une porte en n'utilisant qu'une seule main

Dans cet exemple, le robot tient un objet dans sa main gauche. Nous avons modifié la documentation de la porte pour n'autoriser que les prises de la porte par la main droite. La nouvelle documentation de la porte est présentée en figure 7.7.
La figure 7.8 présente un chemin solution pour le passage de porte en n'utilisant qu'une seule main. Le robot est obligé d'ouvrir la porte, de la lâcher, de passer de l'autre côté puis de fermer la porte.

7.5.3 Passage d'une porte coulissante

Dans cet exemple, la documentation de la porte est toujours celle présentée en figure 7.1, mais le degré de liberté de la porte est en translation et non pas en rotation. Un chemin solution à ce problème est présenté en figure 7.9.

Figure 7.6 – Trajectoire de marche à l'équilibre dynamique pour le robot Partner qui passe une porte. Une chaise rend le problème de planification plus difficile.

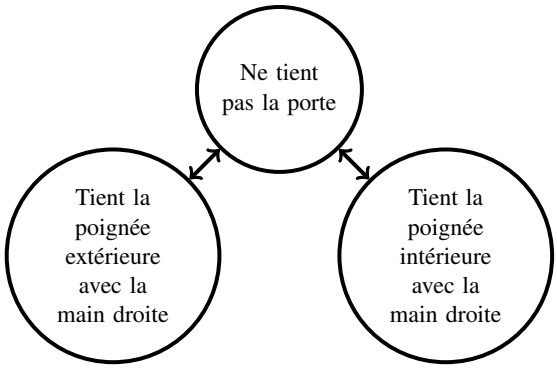


Figure 7.7 – Documentation d'une porte : graphe des contraintes appliquées au robot lors du passage d'une porte en utilisant ses deux mains.

7.6 Conclusion et perspectives

7.6.1 Contribution

Dans ce chapitre, nous avons proposé un formalisme pour la planification de tâches de manipulation complexes. Par complexe, nous entendons que la tâche ne peut pas être exprimée comme une unique pile de tâches de cinématique inverse. La représentation du

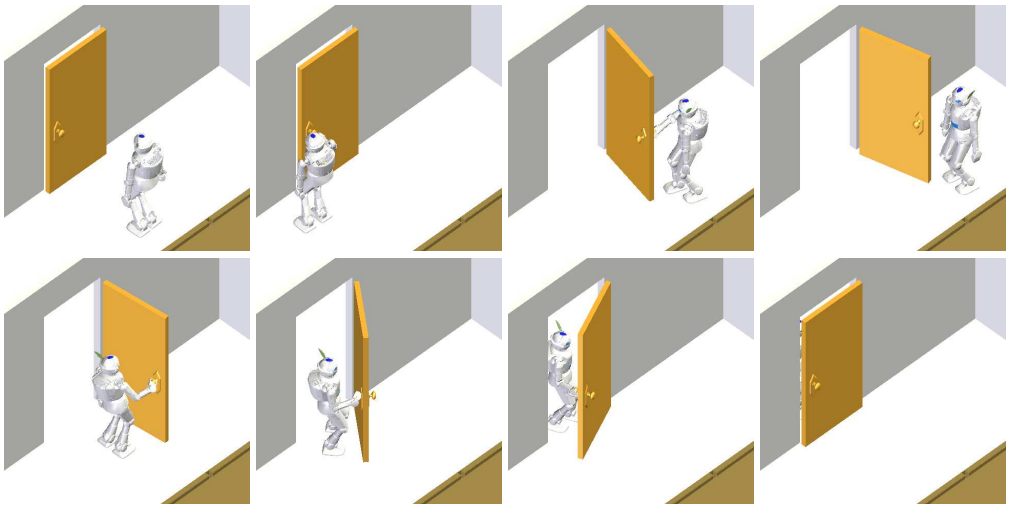


Figure 7.8 – Passage d'une porte en n'utilisant que la main droite.

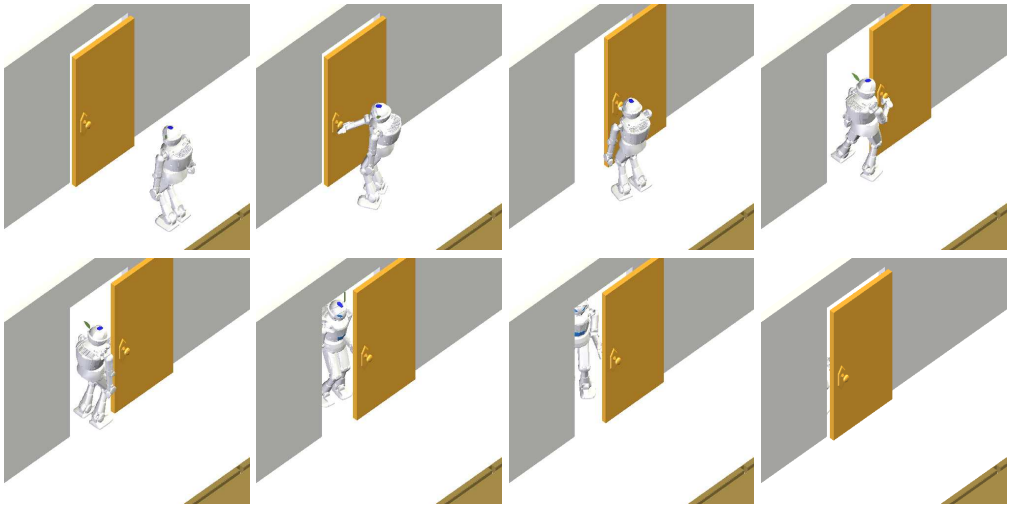


Figure 7.9 – Passage d'une porte coulissante.

système (robot,objet) proposée permet de planifier des mouvements au cours desquels les contraintes appliquées au robot varient.
Nous avons illustré ce formalisme par plusieurs exemples d'ouvertures de portes, comprenant des documentations différentes.

7.6.2 Perspectives

Ce travail a été réalisé en partant du point de vue de la planification de mouvement, et en formalisant les informations dont un algorithme de planification a besoin pour réaliser des mouvements au cours desquels les contraintes varient. La limite principale de cette présentation est qu'elle nécessite qu'un utilisateur définisse la documentation

de chaque objet, voire de chaque problème, comme dans l'exemple où le robot a la main gauche inutilisable. Ce travail pourrait être automatisé et généralisé en utilisant des descriptions fonctionnelles des objets, ainsi que des capacités de manipulation des robots, afin d'inférer et de construire les documentations présentées ici.

Chapitre 8

Conclusion et perspectives

8.1 Contribution

Le travail présenté dans cette thèse propose quelques contributions à la planification de mouvement randomisée pour les systèmes hautement dimensionnés que sont les acteurs digitaux et les robots humanoïdes.
Ces quinze dernières années, le paradigme simplificateur de la recherche aléatoire de chemins a permis à la planification de mouvement de considérer des systèmes de plus en plus complexes, que ce soit du point de vue géométrique ou cinématique. Notre travail a consisté à choisir des outils pour rendre plus efficace cette recherche, moins aléatoire souvent, en essayant de conserver ses propriétés de généricité. Les outils en question proviennent de domaines divers : techniques statistiques de réduction de dimension, cinématique inverse généralisée, propriétés de commandabilité.
Plus précisément, nos contributions portent sur :
– l'accélération des méthodes d'échantillonnage dans les espaces très contraints par l'utilisation de la réduction de dimension linéaire,
– l'utilisation de méthodes de cinématique inverse pour la planification sous contraintes en robotique humanoïde,
– la planification de trajectoires de marche sans collision, ainsi que les propriétés de commandabilité en temps petit des robots humanoïdes,
– une représentation des objets manipulables qui permet la réalisation de tâches de manipulation complexes, dépassant le formalisme usuel de la cinématique inverse.

8.2 Perspectives

Certaines des expériences présentées dans cette thèse ont été validées sur une plateforme réelle. Toutefois, la principale perspective de ce travail est la meilleure intégration des méthodes développées dans un système robotique. Les sorties de nos algorithmes pour

l'instant sont des suites de configurations, décrivant de façon plus ou moins fine une trajectoire sans collision à même de résoudre une tâche passée en entrée. Il ne s'agit pas de l'information la plus pertinente lors de l'intégration sur un robot.
En effet, la mise en pratique des algorithmes de planification nécessite souvent la prise en compte d'erreurs, qu'elles soient liées à la perception du monde ou à l'exécution d'action par le robot. Ces erreurs peuvent être rattrapées au niveau de la planification ou au niveau du contrôle, en fonction de l'influence qu'elles ont sur la réalisation de la tâche.
La réalisation d'un geste d'atteinte vers une poignée de porte, par exemple utilisera probablement une boucle d'asservissement visuel, qui peut être implémentée au niveau du contrôleur du robot, plutôt que du planificateur. Le calcul exact du mouvement articulaire complet du robot lors de la planification est inutile, et ne correspond pas au mouvement exécuté. À l'inverse, une erreur dans la localisation d'un robot humanoïde pendant la locomotion peut nécessiter un appel aux algorithmes de planification pour calculer une nouvelle trajectoire de marche sans collision.
La planification doit donc bien générer des ordres à passer au contrôleur, mais ces ordres ne sont pas nécessairement des positions articulaires, ils peuvent consister en des tâches, de saisie, de posture, d'asservissement visuel, *etc*. Un couplage plus efficace entre planification et contrôle est indispensable pour intégrer sur des robots réels les méthodes puissantes de planification présentées ici.

Références

Berenson, D., Srinivasa, S.S., & Kuffner, J. 2011. Task Space Regions : A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*.

Berenson, Dmitry, Srinivasa, Siddhartha S., Ferguson, Dave, & Kuffner, James J. 2009 (May). Manipulation planning on constraint manifolds. *Pages 625–632 of : Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*.

Brand, Matthew. 2002. Incremental Singular Value Decomposition of Uncertain Data with Missing Values. *Pages 707–720 of :* Heyden, Anders, Sparr, Gunnar, Nielsen, Mads, & Johansen, Peter (eds), *Computer Vision — ECCV 2002*. Lecture Notes in Computer Science, vol. 2350. Springer Berlin / Heidelberg.

Bunch, James R., & Nielsen, Christopher P. 1978. Updating the singular value decomposition. *Numerische Mathematik*, **31**, 111–129.

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

Chapuis, A., & Droz, E. 1949. *Les automates : figures artificielles d'hommes et d'animaux. Histoire et technique*. Neuchâtel, Griffon editions.

Chestnutt, J., Lau, M., Cheung, G., Kuffner, J., Hodgins, J., & Kanade, T. 2005. Footstep planning for the Honda ASIMO humanoid. *Pages 629–634 of : Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W., Kavraki, L.E., & Thrun, S. 2005. *Principles of Robot Motion : theory, algorithms, and implementation*. MIT Press.

Cortes, J., Simeon, T., & Laumond, JP. 2002. A random loop generator for planning the motions of closed kinematic chains using PRM methods. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, **2**.

El Khoury, A., Taix, M., & Lamiraux, F. 2011. Path Optimization for Humanoid Walk Planning : an Efficient Approach. *to appear in Int. Conf. Informatics in Control, Automation and Robotics (ICINCO) 2011*.

Erdogmus, Deniz, Rao, Yadunandana N., Peddaneni, Hemanth, Hegde, Anant, & Principe, Jose C. 2004. Recursive Principal Components Analysis Using Eigenvector Matrix Perturbation. *EURASIP Journal on Applied Signal Processing*, **2004**(13), 2034–2041.

Escande, A., Kheddar, A., Miossec, S., & Garsault, S. 2009. Planning support contact-points for acyclic motions and experiments on HRP-2. *Pages 293–302 of : Experimental Robotics*. Springer.

Esteves Jaramillo, C. 2007. *Motion planning : from digital actors to humanoid robots*. Ph.D. thesis, Institut National Polytechnique, Toulouse, 100p. Doctorat.

Ferre, E., & Laumond, J.P. 2004. An iterative diffusion algorithm for part disassembly. *Pages 3149–3154 of : 2004 International Conference on Robotics and Automation (ICRA'2004)*.

Goodman, J.E., & O'Rourke, J. 2004. *Handbook of discrete and computational geometry*. CRC press.

Han, L., & Amato, N.M. 2000. A kinematics-based probabilistic roadmap method for closed chain systems. *Pages 233–245 of : Algorithmic and Computational Robotics : New Directions. The Fourth Workshop on the Algorithmic Foundations of Robotics*.

Hotelling, H. 1933. Analysis of a Complex of Statistical Variables into principal components, J. Ed. *Psychology*, **24**, 417–441.

Hsu, D. 2000 (May). *Randomized Single-query Motion Planning in Expansive Spaces*. Ph.D. thesis, Dept. of Computer Science, Stanford University, Stanford, CA.

Hsu, D., Latombe, J.C., & Motwani, R. 1999. Path planning in expansive configuration spaces. *Int. J. Computational Geometry & Applications*, **9**(4-5), 495–512.

Hsu, D., Latombe, J.C., & Kurniawati, H. 2006. On the Probabilistic Foundations of Probabilistic Roadmap Planning. *The International Journal of Robotics Research*, **25**(7), 627.

Jolliffe, I. T. 2002. *Principal Component Analysis*. Springer.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., & Hirukawa, H. 2003. Biped walking pattern generation by using preview control of zero-moment point. *Pages 1620–1626 of : IEEE International Conference on Robotics and Automation*, vol. 2. Citeseer.

Kanehiro, F., Hirukawa, H., Kaneko, K., Kajita, S., Fujiwara, K., Harada, K., & Yokoi, K. 2004. Locomotion planning of humanoid robots to pass through narrow spaces. *Pages 604–609 of : Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1. IEEE.

Kanehiro, F., Lamiraux, F., Kanoun, O., Yoshida, E., & Laumond, JP. 2008a. A Local Collision Avoidance Method for Non-strictly Convex Polyhedra. *In : 2008 Robotics : Science and Systems Conference*.

Kanehiro, F., Suleiman, W., Lamiraux, F., Yoshida, E., & Laumond, J.P. 2008b. Integrating Dynamics into Motion Planning for Humanoid Robots. *Pages 660–667 of : IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*.

Kaneko, K., Kanehiro, F., Kajita, S., Hirukawa, H., Kawasaki, T., Hirata, M., Akachi, K., & Isozumi, T. 2004. Humanoid robot HRP-2. *Pages 1083–1090 of : Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 2. IEEE.

Kanoun, O. 2009. *Contribution à la planification de mouvement pour robots humanoïdes*. Ph.D. thesis, Institut National des Sciences Appliquées, Toulouse.

Kanoun, O., Lamiraux, F., & Wieber, P.B. 2009. Prioritizing linear equality and inequality systems : application to local motion planning for redundant robots. *In : IEEE International Conference on Robotics and Automation (ICRA 2009)*. 6 pages.

Kanoun, Oussama, Laumond, Jean-Paul, & Yoshida, Eiichi. 2010. Planning Foot Placements for a Humanoid Robot : A Problem of Inverse Kinematics. *The International Journal of Robotics Research*, 0278364910371238.

Karaman, S., & Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. *Proceedings of Robotics : Science and Systems, Zaragoza, Spain*.

Kavraki, LE, Svestka, P., Latombe, J.C., & Overmars, MH. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, **12**(4), 566–580.

Khatib, O., Sentis, L., Park, J., & Warren, J. 2004. Whole body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, **1**(1), 29–43.

Kuffner, J., & LaValle, S. 2000 (Apr.). RRT-Connect : An efficient approach to single-query path planning. *In : Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA*.

Latombe, J.C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

Laumond, J.-P., Jacobs, P.E., Taix, M., & Murray, R.M. 1994. A motion planner for nonholonomic mobile robots. *Robotics and Automation, IEEE Transactions on*, **10**(5), 577 –593.

Laumond, J.P. 2006. Kineo CAM : a success story of motion planning algorithms. *Robotics & Automation Magazine, IEEE*, **13**(2), 90–93.

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

Lozano-Perez, Tomas. 1983. Spatial Planning : A Configuration Space Approach. *IEEE Transactions on Computers*, **32**(2), 108–120.

Nakamura, Y., & Hanafusa, H. 1986. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME, Transactions, Journal of Dynamic Systems, Measurement, and Control*, **108**, 163–171.

Oriolo, G., & Vendittelli, M. 2009. A control-based approach to task-constrained motion planning. *Pages 297–302 of : Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE.

Pearson, K. 1901. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, **2**(6), 559–572.

Perrin, N., Stasse, O., Lamiraux, F., & Yoshida, E. 2011. A Biped Walking Pattern Generator based on "Half-Steps" for Dimensionality Reduction. *Robotics and Automation, 2011. Proceedings. ICRA'11. IEEE International Conference on*.

Pettré, J., Laumond, J.P., & Siméon, T. 2003. A 2-stages locomotion planner for digital actors. *Page 264 of : Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Ratliff, N., Zucker, M., Bagnell, J.A., & Srinivasa, S. 2009. Chomp : Gradient optimization techniques for efficient motion planning. *Pages 489–494 of : Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE.

Reeds, JA, & Shepp, LA. 1990. Optimal pathsfor a car that goesboth forwards and backwards. *Pacific Journal of Mathematics*, **145**(2).

Roweis, Sam T., & Saul, Lawrence K. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, **290**(5500), 2323–2326.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., & Fujimura, K. 2002. The intelligent ASIMO : System overview and integration. *Pages 2478–2483 of : Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. Ieee.

Schoelkopf, B., Smola, A.J., & Mueller, K.R. 1997. Kernel Principal Component Analysis. *Lecture notes in Computer Science*, 583–588.

Schwartz, J.T., Sharir, M., & Hopcroft, J.E. 1987. *Planning, geometry, and complexity of robot motion*. Ablex Publishing Corporation.

Sekhavat, S., & Laumond, J.-P. 1998. Topological property for collision-free nonholonomic motion planning : the case of sinusoidal inputs for chained form systems. *Robotics and Automation, IEEE Transactions on*, **14**(5), 671 –680.

Siciliano, B., & Khatib, O. 2008. *Springer handbook of robotics*. Springer-Verlag New York Inc.

Siméon, T., Laumond, J.P., & Nissoux, C. 2000. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, **14**(6), 477–493.

Simeon, T., Laumond, J.P., Cortes, J., & Sahbani, A. 2004. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, **23**(7-8), 729.

Stilman, M. 2007. Task constrained motion planning in robot joint space. *Pages 3074–3081 of : Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*. Citeseer.

Takagi, S. 2006. Toyota partner robots. *Journal-Robotics Society of Japan*, **24**(2), 62.

Tenenbaum, Joshua B., Silva, Vin de, & Langford, John C. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, **290**(5500), 2319–2323.

Vendittelli, M., Oriolo, G., Jean, F., & Laumond, J.-P. 2004. Nonhomogeneous nilpotent approximations for nonholonomic systems with singularities. *Automatic Control, IEEE Transactions on*, **49**(2), 261 – 266.

Vukobratović, M., Borovac, B., Surla, D., & Stokić, D. 1990. *Biped Locomotion : dynamics, stability, control, and application*. Springer-Verlag.

Wall, M., Rechtsteiner, A., & Rocha, L. 2003. Singular value decomposition and principal component analysis. *A practical approach to microarray data analysis*, 91–109.

Wieber, Pierre-Brice. 2002. On the stability of walking systems. *In : Proceedings of the International Workshop on Humanoid and Human Friendly Robotics*.

Yakey, J.H., LaValle, S.M., & Kavraki, L.E. 2001. Randomized path planning for linkages with closed kinematic chains. *Robotics and Automation, IEEE Transactions on*, **17**(6), 951–958.

Yoshida, E., Belousov, I., Esteves, C., & Laumond, J.-P. 2005 (5-5). Humanoid motion planning for dynamic tasks. *Pages 1 –6 of : Humanoid Robots, 2005 5th IEEE-RAS International Conference on*.

Yoshida, E., Kanoun, O., Esteves, C., & Laumond, J.P. 2006. Task-driven support polygon reshaping for humanoids. *Pages 208–213 of : Humanoid Robots, 2006 6th IEEE-RAS International Conference on*.

Yoshida, E., Mallet, A., Lamiraux, F., Kanoun, O., Stasse, O., Poirier, M., Dominey, P.F., Laumond, J.P., & Yokoi, K. 2007. "Give me the purple ball"-he said to HRP-2 N. 14. *Pages 89–95 of : Humanoid Robots, 2007 7th IEEE-RAS International Conference on*. IEEE.

Zwald, Laurent, & Blanchard, Gilles. 2005. On the Convergence of Eigenspaces in Kernel Principal Component Analysis. *In : Neural Information Processing Systems*.